Prediction of Netflix Movie Revenues

Mini Project - Part 1

Ramesh Manian rmanian@stanford.edu
Uzair Mansuri umansuri@stanford.edu

1 Abstract

In this project, we try to address the challenge of predicting Gross movie revenues for popular video streaming network Netflix, based on multiple features extracted from the dataset. This type of analysis can be useful for video streaming networks to asses which type of movies should be considered to increase popularity of streaming services.

1.1 Dataset description

'movies merged' dataset used in our Project was compiled by Dr. Guy Lebanon, an ex-professor at Georgia Tech. The dataset is a collection of 40,789 rows of media titles, whose details include name, running time, budget, revenue and other important information about the media encapsulated in 39 covariates. The data contained in it is real data collected by querying IMDb's API and joining it, movie title being the join key, with a separate dataset of movie budgets and gross earnings, whose source is unknown to us. The data seems comprehensive and accurate. We randomly sampled a few movie titles and checked for its accuracy on the net using publicly available information. We are satisfied with the veracity of information presented in the dataset.

We were both excited by the dataset because, this is a treasure trove of information about a subject that has consumed much of our lives. We are excited by the prospect of learning what makes a movie a hit or a flop, or what factors might go into affecting the revenue a make means? Do big budgets in movie making guarantee a profit? Perhaps, even more importantly, we found this data was presented in a very messy manner. This meant that we will spend a lot of time cleaning the data and in the process, learning about ways to clean data in R language techniques and syntax.

1.2 Data Cleaning & Features analysis

In the dataset, we found several covariates that are likely irrelevant to predicting the revenue using linear regression techniques that we undertake in this project. For example, the "Plot" of the movie described in text, will not help us in better prediction of revenue. Similarly, covariates such as Poster, Website, imdbID, tomoatoImage, tomatoConsensus, BoxOffice, and tomatoURL also were found to be not useful. Domestic Gross is the amount of revenue a movie earned within the US. Unsurprisingly, it is very highly correlated with Gross and is in fact equal to it for movies that were not released globally. Hence, we removed that for our model consideration.

Our data has a lot of NA values. For instance, 'BoxOffice' column in the dataset has all NA values. Most importantly, out of 40,789 observations, 36,206 observations have missing values for Gross and 'Budget'. Since we cannot use any observations with missing 'Gross', we removed all those rows. This left us with a much diminished, but usable dataset with 4583 observations. There were a few hundred NAs in imdbRating, tomatoRating, Runtime, tomatoMeter, tomatoUserRating and tomatoFresh. We replaced these NAs with the mean value. After running different models, we did not find that replacing NAs with mean value changed the R Squared value. However, the RMSE reduced by a couple of million dollars and we persisted those dataset changes for our follow-up models.

Runtime column in the dataset features string values which has been processed to numeric value (in minutes). The column Genre also represents a list of genres associated with the movie in a string format. We've decided to parse each text string into a binary vector with 1's representing the presence of a genre and 0's the absence.

Released column in the dataset is string representation of year of movie released, which we used to compare with Year column to find and eliminate mis-matches in the data. In addition to the above, we decided to drop the Domestic Gross column altogether for modeling purposes, since it basically indicates the amount of revenue a movie earned within the US. Quite likely, it is very highly correlated with Gross and is in fact equal to it for movies that were not released globally.

Data Patterns

Examining the runtime distribution over the years, it appears that Runtime seems to be increasing over the decades with the exception of 1930s and 2000's. No such anamoly exists in the Runtime distribution over the budget categories. Runtime seems to increase from as budget categories go from very low to very high.

As the wins and nominations increase, the gross revenue increases substantially as well. Unsurpris-

ingly, wins and nominations have a linear relationship.

Continuous Response Variable

We would like to predict the 'Gross' revenue of a movie. Revenue of a movie depends upon many variables such as the cost of the movie, its running time, critic and user reviews, which are available to us. We want to understand what covariates are most useful in determining the revenue. Predicting the value of an outcome variable using Linear regression, requires the variable to be numeric in nature. 'Gross' is a numeric quantity whose mean is \$89.6M, median of \$29.19M and a third quintile of \$99.75M. The range of revenue is between \$0M and \$2.78B suggesting that there are many outliers, as is shown by box plot. This makes it a very interesting variable to predict and will likely have a high MSE.

Binary Response Variable

One of the questions often asked in the context of movies; is the movie any good? While the answer may be subjective, in the context of data analysis, it is an important question. One of the variables in our dataset is 'IMDB Rating'. This is a rating given by IMDB users for movies. We would like to predict that using other data in the dataset. IMDB Rating is a score between 1 and 10. Instead of predicting a numeric score, we would like to classify a movie as "Hit" or "Miss" based on a threshold score. If the predicted score is equal to or above 7, our binary classifier will label it "Hit", else a "Miss". "Hit" equals "1" and "Miss" equals "0".

1.3 Correlation Analysis

To get a sense of correlation between our covariates and the influence of each covariate on our response variable, we decided to plot the correlation matrix as shows in figure 1 below. We can clearly see correlation of clusters forming around the diagonal, showing those group of covariates as highly correlated. As an example, `imdbRating`, `imdbVotes` and `tomatoRating` show strong positive correlation, which makes sense since both are ratings given by consumers on different platforms. There are some interesting associativity patterns within data highlighted in correlation matrix, for instance strong negative correlation between `Gross` and `Budget`. This association makes sense since the overall Gross revenue will be less, higher the total Budget of the movie is. Also, one interesting negative correlation is between `tomatoRating` and `tomatoRotten`, which suggests that higher "Rotten" ratings will negatively impact overall `tomatoRating`.

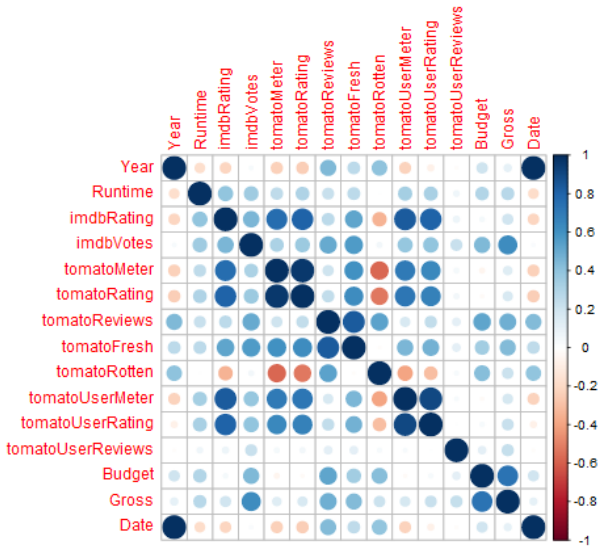


Figure 1: Correlation Matrix

2 Prediction

We attempt to create a model that will best predict the `Gross` revenue. Our strategy for building a reasonable regression model for predicting our numerical outcome variable, `Gross` is a tiered approach. As we go up the tier, we typically change the nature of covariates and examine the results from that model. The following sections capture our effort in each of the tier and a tabulation of what we found.

2.1 Evaluation

Numerical Covariates

We explored with using only numeric covariates. We did two experiments in this. We started with using most of all the numeric covariates available in the dataset, 15 of them to get a baseline. We then examined the correlation plot mentioned above and picked out only `Runtime`, `imdbRating`, `imdbVotes`, `tomatoReviews`, `tomatoFresh`, `tomatoRating`, and `Budget`. In experiment 1, we got $R^2 = 0.6838$, $RMSE = \$98,119,611$ and error from Cross Validation of $\$99,373,484$. In experiment 2, we got an $R^2 = 0.6773$ and $RMSE = \$96,045,332$. This represents a 2.11% decrease in RMSE.

Transformation of Numerical Variables

We approached transformation by first plotting the residuals vs fitted values, which helped us in identifying non-linear relationships. This is because of the

fact that the plot exhibited a pattern where Residuals somewhat increased as `Gross` increased. If there were no nonlinear relationships, the plot would have looked very scattered with no discernible pattern. To get a better and a deeper understanding of which variables have a correlation with Gross and how it is related, we created a pairwise plot of all features, eliminating NA samples and samples where Gross is zero. This table of plots show that `imdbVotes`,`imdbRating` and `imdbUserRating` possibly have a nonlinear, a power relationship with `Gross`. Also `RuntimeLog` in our trials showed a better correlation to Gross than non-transformed `Runtime`. We also chose `imdbVotes` and `Budget` to be transformed into binary high-low from a common-sense thinking where most people classify a movie as "see" vs "skip" and as "big budget" or "indie" types. So, for doing the transforms, we chose

- imdbVotesBin : binary xform
- BudgetBin : binary xform
- RuntimeLog : log xform
- imdbRatingP2 : power transform (squared)
- imdbVotesP2 : power transform (squared)
- imdbUserRatingP2 : power transform (squared)

With this model, we got $R^2 = 0.6992$, $RMSE = \$95,707,770$ and error from Cross Validation of $\$97,584,745$. This represents a 3.23% increase in R^2, and a further decrease of 0.35% in RMSE from optimized numerical model.

Categorical Variables

Out of the 23 non-numerical variables that were in the dataset, we dropped all but the following: `Genre, Director, Actors, Language, Country, Production`. Our decision was based on less number of factor levels being available in the covariates and missing values in each. A category variable such as `Genre`" may be represented as a set of genres in a data element. For example, the very first data element in our training data is the movie, "American Heist", which has a list of 3 genres: Action, Crime, Drama. Our goal is to reform the dataset in such a way that all the Genres (and other categorical variables) are represented as independent features. We did this using **one-hot encoding** technique, to turn the categorical covariates into binary with value of 1 for the genre listed.

RMSE for this model was $\$154,925,828$ Compare this to $\$94,562,545$. The error is BIG. The average R Squared across all the different training sample sets was very low at **0.1778**. While the above stats don't mean that the model with only the categorical variables was "bad", it certainly shows that this is not practical to apply the model without further calibration.

Interaction Variables

In this section, our goal was to see how we can add new covariates, that interact with each other like `imdbRating` and `imdbVotes` can be included to better train the model. To that end, we created a correlation plot of numerical variables and we came up with a set of interactions involving a combination of the numerical variables. We believe that we could have been a bit more choosy and could have experimented more and whittled down the combinations. This experiment was done in two parts; first one with all possible interactions of covariates, but without transformations. The second one used the dataset with transformed variables, and a slightly smaller set of numerical variables. For instance, we removed `Year` from consideration. It is important to note that we got excellent results: $R^2 = 0.8005$, $RMSE = \$77,935,567$ $CV = \$89,203,508$. For the second experiment, we got : $R^2 = 0.7732$, $RMSE = \$83,096,040$ $CV = \$88,082,565$. While the first experiment was quite better performing than the second one, we consider both to be high performing.

Lasso

After standardizing the covariates and the response variable, we applied Lasso Regression, with varying penalty threshold, to see if we could get an improvement over OLS on the CV error. We computed an optimum λ value of 0.01. Adding penalty thresholds in Lasso did not seem to improve accuracy of our prediction, most likely because we have already selected and removed irrelevant covariates from our dataset.

3 Classification

In this section, we will be modeling to classify whether a particular movie is a "Hit" or "Miss" based on `imdbRating`. We define a threshold such that if the total `imdbRating` is above the stated threshold, the movie is "Hit", otherwise it is not. The threshold is set above the median rating as 7.

3.1 Evaluation

By choosing the threshold for `imdbRating`, we now have a balanced dataset with binary outcome variable (1 for `imdbRating` > 7, 0 otherwise). For the baseline model, we decided to run a basic logistic regression with intercept only, named as "Null" model. For feature selection, we used forward stepwise selection in order to determine the best covariates. Forward stepwise selection found that the model with the lowest AIC is one that contains all covariates in our dataset. The selected model gives us 0-1 loss of 0.0731 and

cross-validation error of 0.0740.

We also computed confusion matrix for our selected model as shown in Table 1 below, which indicates our model represents a classifier with Accuracy of 91.52%.

Table 1: Confusion Matrix for selected Model

		Predicted		
		0	1	Total
Actual	0	2253	229	2482
	1	49	749	798
	Total	2302	978	3280

The ROC curve for our selected model on training set, provides an AUC of 0.97, which is very close to 1, indicating a robust classifier, as shown in figure 2. We also computed the AUC of 0.5 on our baseline "Null" model which indicates our selected model is more reasonable than baseline, considering no change to threshold for classification.

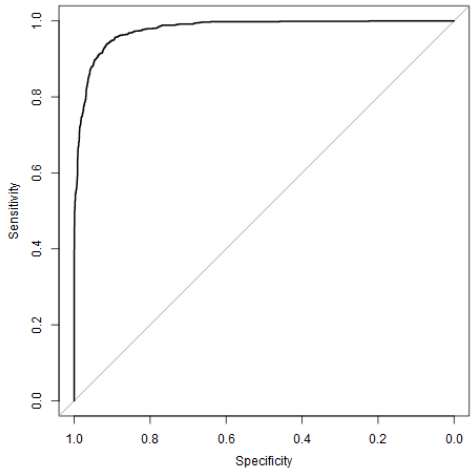


Figure 2: ROC Curve

3.1.1 Interaction with covariates

We also evaluated our model by adding interaction within the covariates in our dataset in order to provide more options for the model to describe population model based on the training data. With the interaction model, we get a lower 0-1 loss of 0.0603 on the training set and cross validation error of 0.0740. The lower training error as compared to cross-validation error is a sign of higher variance than the preceding model. Also, our estimate of cross-validation error i.e. the estimate of the test error, is almost identical as our selected model with all covariates. This indicates that the bias and variance did not change by using additional interaction terms.

4 Summary

- One of the most important task when it comes to analyzing data is to visualize the data so that we can get an overall story that the data is trying to tell us. Simple scatter plots, and correlation plots will teach us the significance of features. Picture is worth a thousand words or lines of code in this case
- When data are being used in training, sampling the data between iterations is crucial in eliminating biases
- Learning the nature of features such as numeric and categorical will help us understand the models and prediction methods that we can use
- Not all features or data points that we gather are important. We have to be able to look at the data, its plots and eliminate features that don't improve our prediction model
- We have to be very careful to not overfit the model, as this will result in low performing predictions. It is also important to not underfit the model. Hence the importance of balancing bias and variance. Overfitted model will have high variance in generalization error whereas underfitted models are likely to have low variance. This means that complex models will perform well in training, but are inconsistent in test conditions. The opposite is true is very less complex model. Underfit models don't learn the data well, where overfit models learn the training data too well and when unseen new samples are shown, it can perform poorly. The goal is to strike the sweet spot, **"Goldilocks Principle"**

We intend to apply our best models for classification and regression on the held out test set and compare the results with our generalization error, i.e. the estimate of the test error from cross validation. With lessons learned on refining dataset, calibrating the models for both regression and classification task for "best fit" to the population model, we aim to use our model that we trained using training data, to predict the Gross revenue of the movie and whether the movie is a "Hit" or not on the holdout test set, in the next part of our Project.

5 References

- **Dataset**: https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6242/project/movies_merged
- **MS&E 226 Lecture Notes**: https://web.stanford.edu/class/msande226/l_notes.html
- **R for Data Science**: https://r4ds.had.co.nz/
- **Cookbook for R**: http://www.cookbook-r.com/
- **Cross Validated**: https://stats.stackexchange.com/

APPENDIX

MS&E 226: PR1

Ramesh Manian (rmanian@stanford.edu) & uzair Mansuri (umansuri@stanford.edu)

10/19/2020

Project group members: *Ramesh Manian, Uzair Mansuri.*

Set seed for repeatability

```
set.seed(3945827)
```

Load the dataset and explore

```
load('movies_merged')
cat("movies-merged Dataset has", dim(movies_merged)[1], "rows and", dim(movies_merged)[2], "columns", en
```

```
## movies-merged Dataset has 40789 rows and 39 columns
```

Let us get the data into a dataset and get its column names

```
df = movies_merged
cat("Column names:", end="\n", file="")
```

```
## Column names:
```

```
colnames(df)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Domestic_Gross"    "Gross"             "Date"
```

Cleaning up data

Remove non-movie rows

```
# Remove all rows from df that do not correspond to movies
df2 <- df[df$Type == "movie",]
dim(df2)
```

```
## [1] 40000    39
```

```
# save df2 as df
df <- df2

#rem_cols <- c(tomatoConsensus, Plot, Poster, Website, imdbID, tomoatoImage, tomatoConsensus, BoxOffice
```

Drop rows with missing `Gross` value

our goal is to model `Gross` revenue against other variables, therefore, rows that have missing `Gross` values are not useful to us.

```
# Remove rows with missing Gross value
# subset rows with valid gross values
df_gross_val = subset(df, !is.na(df$Gross))
df = df_gross_val
cat("Movies only dataset with valid Gross value has", dim(df)[1], "samples and", dim(df)[2], "features"
```

```
## Movies only dataset with valid Gross value has 4558 samples and 39 features
```

Process `Runtime` column

The variable `Runtime` represents the length of the title as a string. Let us convert it to a numeric value (in minutes) and replace `Runtime` with the new numeric column.

```
# Replace df$Runtime with a numeric column containing the runtime in minutes
# look for three time patterns in run_time "xx h yy min" or "xx h" or "xx min"
runtime_pattern = c("(\\d+)\\sh\\s(\\d+)\\smin", "(\\d+)\\sh", "(\\d+)\\smin")
normalize_NA_convert_to_min <- function(s) {
    # convert N/A to NA
    if (s == "N/A"|| s == "n/a" || s == "N/a" || s == "n/A") return(NA)
    # go through the possible ways time can be shown
    for (i in 1:length(runtime_pattern)) {
        val = str_match(s, runtime_pattern[i])
        if (!is.na(val[1])) {
            if (i == 1) return(as.numeric(val[2])*60 + as.numeric(val[3]))
            else if (i == 2) return(as.numeric(val[2])*60)
            else if (i == 3) return(as.numeric(val[2]))
        }
    }
}
df$Runtime = sapply(df$Runtime, normalize_NA_convert_to_min)
```

Now investigate the distribution of `Runtime` values and how it changes over years (variable `Year`, which you can bucket into decades) and in relation to the budget (variable `Budget`). Include any plots that illustrate.

```
# Investigate the distribution of Runtime values and how it varies by Year and Budget
str(unique(df$Year))
```

```
##  num [1:92] 2005 2004 2002 1986 1993 ...
```

Since there are too many unique years, let us bucket them in to decades

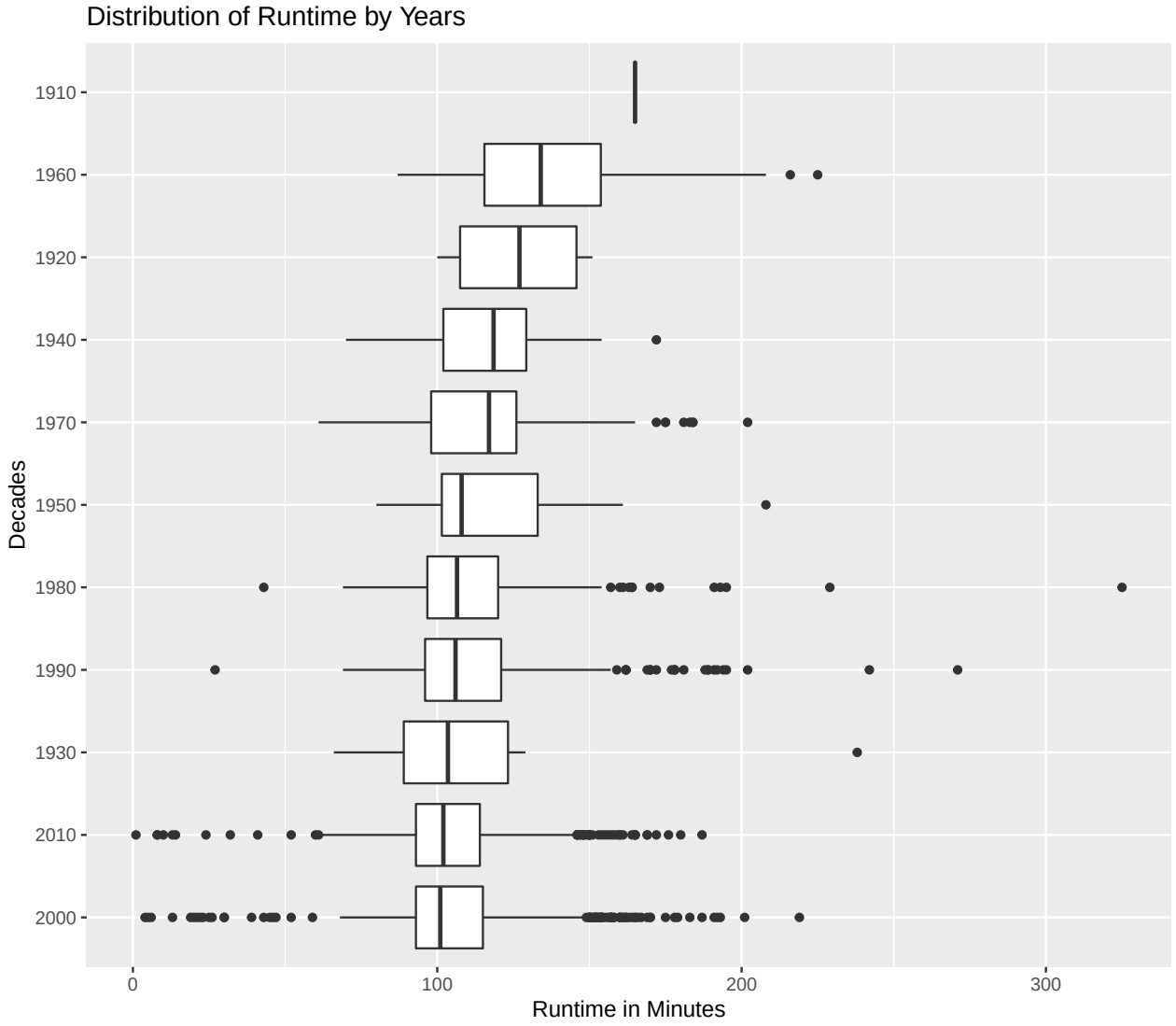
```
# Add a 'Decade' column
df['Decade'] = floor(df$Year / 10) * 10
```

```
# Remove rows that have NA values in runtime, and decade columns
df_rt_decade <- subset(df, !is.na(df$Runtime) &
                       !is.na(df$Decade))
# 39249 observations are there with just Runtime and Decade being non-NA
cat("Number of rows with non NA values in Runtime and decade :",
    nrow(df_rt_decade))
```

```
## Number of rows with non NA values in Runtime and decade : 4520
```

Let us now see how the run time varies over the years by plotting the distribution by a box plot

```
# Q2a. plot boxplots of year vs. runtime by decades
ggplot(df_rt_decade, aes(reorder(Decade, Runtime, median), Runtime)) +
    geom_boxplot() + coord_flip() + scale_x_discrete("Decades") +
    ggtitle("Distribution of Runtime by Years") +
    labs(y="Runtime in Minutes", x="Decade")
```



Let us extract valid observations; ones with budget information in our case

```
# Remove rows that have NA values in runtime and budget columns
df_rt_budget <- subset(df, !is.na(df$Runtime) &
```

```
                      !is.na(df$Budget))
# 4520 observations are there with just Runtime and Budget being non-NA
cat("Number of rows to be removed with non NA values in Runtime and budget :",
    nrow(df_rt_budget))
```

```
## Number of rows to be removed with non NA values in Runtime and budget : 4520
```

Eliminate mismatched rows

Compare the `Released` column (string representation of release date) with either `Year` or `Date` (numeric representation of the year) to find mismatches.

```
# Remove mismatched rows
# Remove all rows with Year/Date/release mismatch
correct_year_mismatch = function(dft){
    dft["YearRel"] = as.numeric(format(as.Date(dft$Released,
                                       format = "%Y-%m-%d"), "%Y"))
    c1 = (is.na(dft$Date) | (dft$Year == dft$Date) |
              (!is.na(dft$YearRel) && dft$YearRel == dft$Date))
    dft = dft[c1,]
    # delete temporary column or numeric representation of Released
    dft = subset(dft, select=-YearRel)
    return(dft)
}

df_non_mismatched_year = correct_year_mismatch(df)
new_cnt = nrow(df_non_mismatched_year)
# compute percentage of mis-matched rows removed
df = df_non_mismatched_year
cat("Non-mismatched dataset has", dim(df)[1], "samples and", dim(df)[2], "features", "\n")
```

```
## Non-mismatched dataset has 4558 samples and 40 features
```

Drop `Domestic_Gross` column

`Domestic_Gross` is basically the amount of revenue a movie earned within the US. Quite likely, it is very highly correlated with `Gross` and is in fact equal to it for movies that were not released globally. Hence, we will remove it for modeling purposes.

```
# Exclude the `Domestic_Gross` column
df_gross_removed = subset(df, select=-Domestic_Gross)
df = df_gross_removed
cat("Domestic_Gross removed dataset has", dim(df)[1], "samples and", dim(df)[2], "features", "\n")
```

```
## Domestic_Gross removed dataset has 4558 samples and 39 features
```

Final preprocessed dataset

Let us look at the dimensions of the preprocessed dataset that we will be using for modeling and evaluation. Let us also print all the final covariates list.

```
# Print the dimensions of the final pre-processed dataset and column names
cat("Pre-processed dataset has", dim(df)[1], "samples and", dim(df)[2], "features. Features are printed
```

```
## Pre-processed dataset has 4558 samples and 39 features. Features are printed below.
```

```
pre_processed_df = df
print(sort(colnames(df)))
```

```
##  [1] "Actors"            "Awards"            "BoxOffice"
##  [4] "Budget"            "Country"           "Date"
##  [7] "Decade"            "Director"          "DVD"
## [10] "Genre"             "Gross"             "imdbID"
## [13] "imdbRating"        "imdbVotes"         "Language"
## [16] "Metascore"         "Plot"              "Poster"
## [19] "Production"        "Rated"             "Released"
## [22] "Response"          "Runtime"           "Title"
## [25] "tomatoConsensus"   "tomatoFresh"       "tomatoImage"
## [28] "tomatoMeter"       "tomatoRating"      "tomatoReviews"
## [31] "tomatoRotten"      "tomatoURL"         "tomatoUserMeter"
## [34] "tomatoUserRating"  "tomatoUserReviews" "Type"
## [37] "Website"           "Writer"            "Year"
```

Model Evaluation

Numeric variables

We will be using Linear Regression to predict `Gross` based on available *numeric* variables.

```
# Numeric
# ========================
# Build & evaluate model 1 (numeric variables only)
# This function randomizes data and splits into train and test datasets
pre_process_data = function(df, test_frac=0.20){
    # shuffle data - sample(nrow(df)) gives randomized indices for selection
    df = df[sample(nrow(df)), ]
    num_samples = nrow(df)
    num_test_samples = as.integer(num_samples * test_frac)
    num_train_samples = num_samples - num_test_samples
    train_data = df[1:num_train_samples, ]
    test_data = df[(num_train_samples+1):num_samples, ]
    return(list(train_data=train_data, test_data=test_data))
}

# get data processed and ready.
rslt = pre_process_data(df)
train_data = rslt$train_data
test_data = rslt$test_data
```

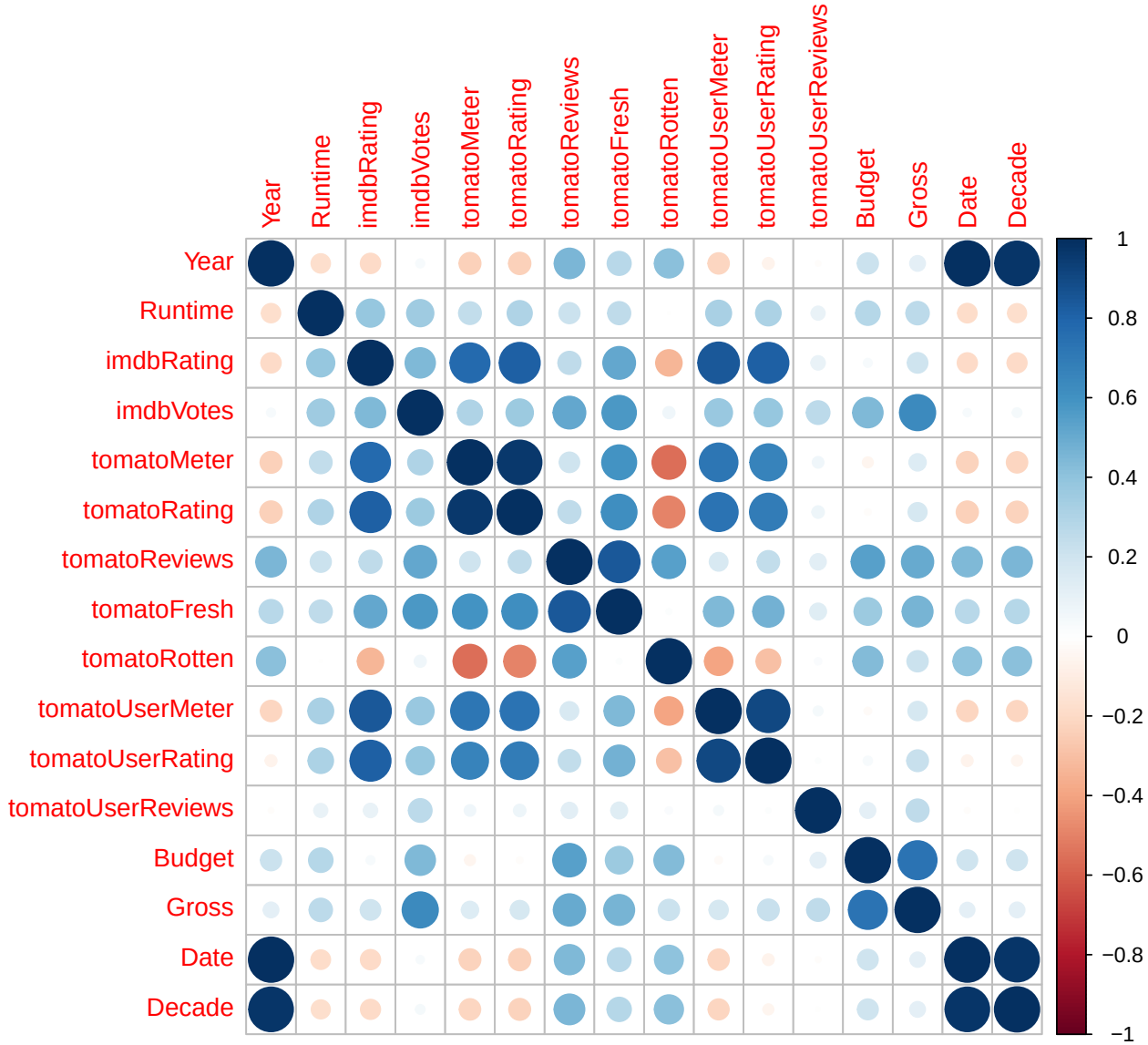
For our prediction, we will focus on numeric variables. Use the filter function which gets all types of numeric variabels including floats and ints.

```
train_data_numeric = Filter(is.numeric, train_data)
test_data_numeric = Filter(is.numeric, test_data)

# replace some of the numeric variables that have NA in their observations with the column mean
train_data_numeric$imdbRating[is.na(train_data_numeric$imdbRating)] <- mean(train_data_numeric$imdbRati
train_data_numeric$Runtime[is.na(train_data_numeric$Runtime)] <- mean(train_data_numeric$Runtime, na.rm
train_data_numeric$tomatoMeter[is.na(train_data_numeric$tomatoMeter)] <- mean(train_data_numeric$tomatol
```

```
train_data_numeric$tomatoRating[is.na(train_data_numeric$tomatoRating)] <- mean(train_data_numeric$toma
train_data_numeric$tomatoReviews[is.na(train_data_numeric$tomatoReviews)] <- mean(train_data_numeric$to
train_data_numeric$tomatoFresh[is.na(train_data_numeric$tomatoFresh)] <- mean(train_data_numeric$tomatoF
train_data_numeric$tomatoUserMeter[is.na(train_data_numeric$tomatoUserMeter)] <- mean(train_data_numeri
train_data_numeric$tomatoUserRating[is.na(train_data_numeric$tomatoUserRating)] <- mean(train_data_nume

# Let us also ensure that the dataset is free from NA values
train_data_numeric_nonNA <- na.omit(train_data_numeric)
test_data_numeric_nonNA <- na.omit(test_data_numeric)
```

We might also want to see if we should use only a subset of numeric variables to better understand the relationship between features. To do this, we could do a correlation plot. This plot will show correlation between -1 and +1. Very strongly positively correlated ones will be shown as very dark blue circle and the intensity of blue will smaller as the +ve correlation decreases. -ve correlation is shown in orange similarly.

```
tr_cor_df = cor(train_data_numeric_nonNA)
corrplot(tr_cor_df, method="circle")
```



Based on the above, I chose only Runtime, imdbRating, imdbVotes, tomatoReviews, tomatoFresh, tomatoRating, and Budget. I will drop the rest of the columns.

```
train_data_numeric_nonNA_subset = subset(train_data_numeric_nonNA,
                              select=-c(tomatoRotten))
test_data_numeric_nonNA_subset = subset(train_data_numeric_nonNA,
                              select=-c(tomatoRotten))

train_data_numeric_nonNA_subset_2 = subset(train_data_numeric,
                              select=-c(Year, Date, tomatoMeter, tomatoRotten,
                                        tomatoUserMeter, tomatoUserReviews,
                                        tomatoRating))
test_data_numeric_nonNA_subset_2 = subset(test_data_numeric,
                              select=-c(Year, Date, tomatoMeter, tomatoRotten,
                                        tomatoUserMeter, tomatoUserReviews,
                                        tomatoRating))
```

Function for performing linear regression using linear model(lm) command and gathering model stats

Let us create a model and evaluate it

```
numericModel<-lm(Gross~., data = train_data_numeric_nonNA_subset)
print(summary(numericModel))
```

```
##
## Call:
## lm(formula = Gross ~ ., data = train_data_numeric_nonNA_subset)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -583369824  -34585868   -2057826   24636878 1360266621
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.388e+09  4.012e+08   3.460 0.000548 ***
## Year              -1.627e+06  4.118e+06  -0.395 0.692842
## Runtime           -4.578e+05  9.853e+04  -4.646 3.52e-06 ***
## imdbRating        -2.705e+07  3.900e+06  -6.937 4.83e-12 ***
## imdbVotes          4.046e+02  1.702e+01  23.765  < 2e-16 ***
## tomatoMeter        7.360e+05  2.629e+05   2.799 0.005151 **
## tomatoRating      -9.392e+06  5.055e+06  -1.858 0.063251 .
## tomatoReviews     -1.671e+05  7.250e+04  -2.304 0.021280 *
## tomatoFresh        3.187e+05  1.015e+05   3.140 0.001707 **
## tomatoUserMeter   -6.087e+04  2.486e+05  -0.245 0.806626
## tomatoUserRating   6.498e+07  9.905e+06   6.560 6.22e-11 ***
## tomatoUserReviews  5.403e+00  5.488e-01   9.845  < 2e-16 ***
## Budget             2.521e+00  5.499e-02  45.847  < 2e-16 ***
## Date               1.356e+06  4.028e+06   0.337 0.736480
## Decade            -4.250e+05  6.080e+05  -0.699 0.484633
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 98340000 on 3262 degrees of freedom
## Multiple R-squared:  0.6838, Adjusted R-squared:  0.6825
## F-statistic: 503.9 on 14 and 3262 DF,  p-value: < 2.2e-16
```

```
numericModel.rmse <- sqrt(mean(numericModel$residuals^2))
cat("Train RMSE is:", numericModel.rmse, "\n")
```

```
## Train RMSE is: 98119611
```

```
#Cross validation
numericModel_cv <- cvFit(numericModel, data = train_data_numeric_nonNA_subset,
                         y = train_data_numeric_nonNA_subset$Gross,
                         K=10)
print(numericModel_cv)
```

```
## 10-fold CV results:
##       CV
## 99373484
```

Try the second subset

```
numericModel2<-lm(Gross~., data = train_data_numeric_nonNA_subset_2)
print(summary(numericModel2))
```

```
##
## Call:
## lm(formula = Gross ~ ., data = train_data_numeric_nonNA_subset_2)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -620074255  -32016617     123242   21507409 1424378038
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.050e+09  3.169e+08   3.314 0.000928 ***
## Runtime          -4.790e+05  8.842e+04  -5.417 6.45e-08 ***
## imdbRating       -1.621e+07  2.595e+06  -6.247 4.66e-10 ***
## imdbVotes         4.319e+02  1.549e+01  27.883  < 2e-16 ***
## tomatoReviews    -2.734e+05  5.819e+04  -4.697 2.73e-06 ***
## tomatoFresh       4.632e+05  6.762e+04   6.849 8.68e-12 ***
## tomatoUserRating  4.053e+07  5.536e+06   7.322 3.00e-13 ***
## Budget            2.506e+00  5.141e-02  48.749  < 2e-16 ***
## Decade           -5.277e+05  1.577e+05  -3.347 0.000826 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 96170000 on 3603 degrees of freedom
##   (35 observations deleted due to missingness)
## Multiple R-squared:  0.6773, Adjusted R-squared:  0.6766
## F-statistic: 945.3 on 8 and 3603 DF,  p-value: < 2.2e-16
```

```
numericModel2.rmse <- sqrt(mean(numericModel2$residuals^2))
cat("Train RMSE for Numeric model 2 is:", numericModel2.rmse, "\n")
```

```
## Train RMSE for Numeric model 2 is: 96045332
```

```
#Cross validation
numericModel2_cv <- cvFit(numericModel2, data = train_data_numeric_nonNA_subset_2,
                         y = train_data_numeric_nonNA_subset_2$Gross,
                         K=10)
print(numericModel2_cv)
```

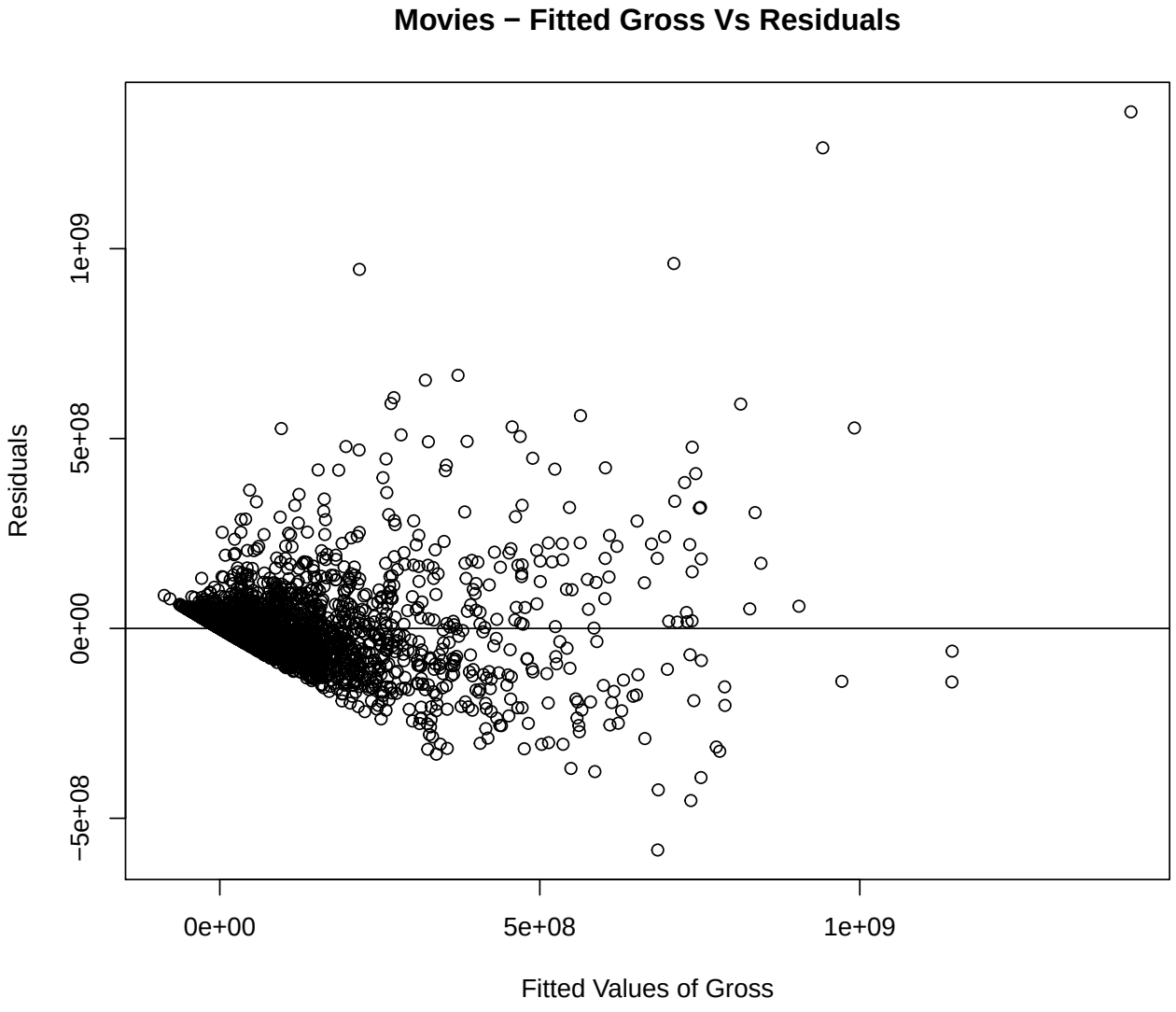
```
## 10-fold CV results:
## CV
## NA
```

Transformations

Let us see if we can improve the prediction quality from basic Numerical variables as much as possible by adding feature transformations of the numeric variables. We will explore both numeric transformations such as power transforms and non-numeric transformations of the numeric variables like binning (e.g. `is_budget_greater_than_3M`).

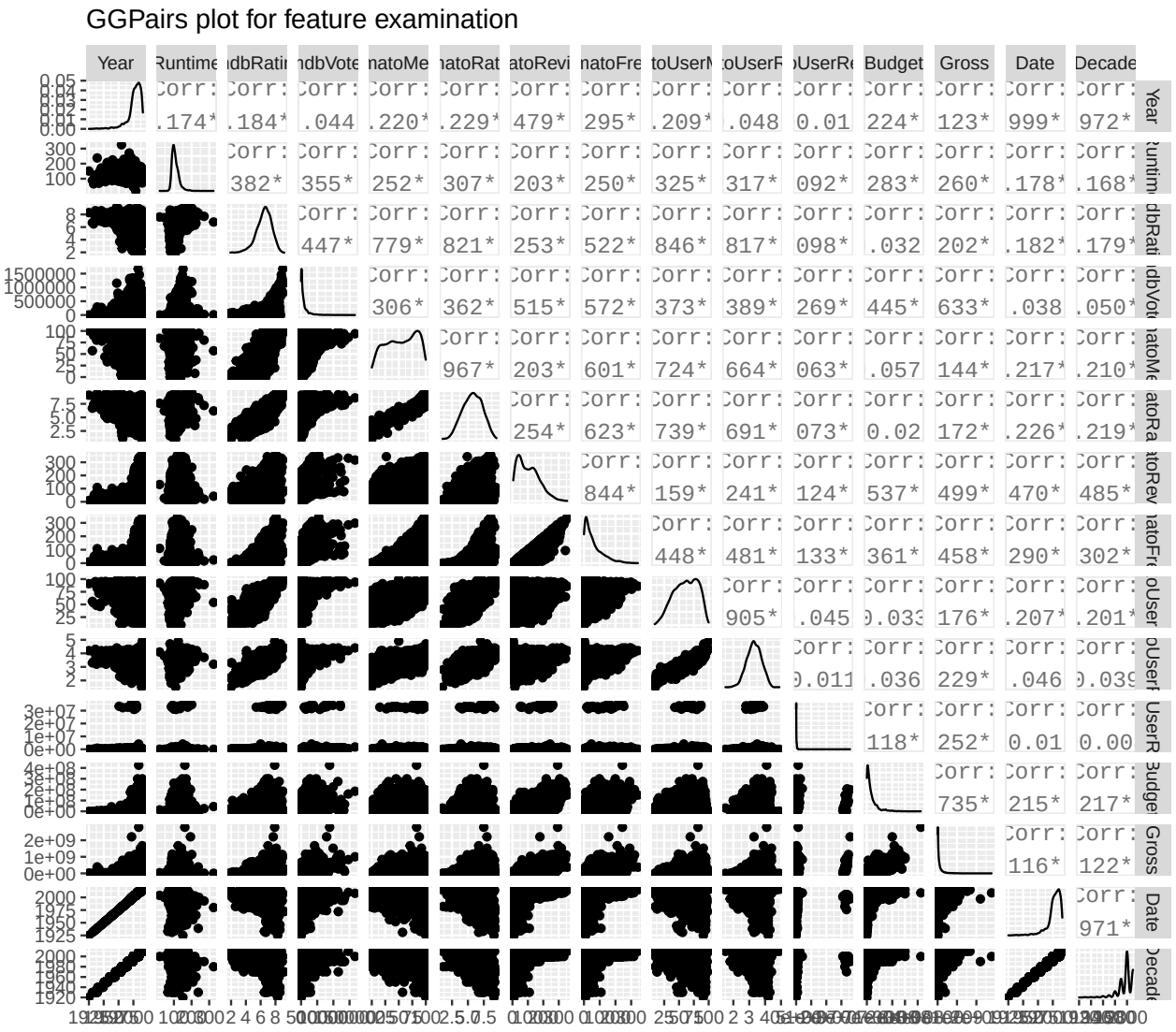
Let us plot the residuals to see if there is non-linearity in features at a gross level. We can do this by plotting residuals for Gross

```
mdl = lm(Gross~., data=train_data_numeric_nonNA_subset)
m.res = resid(mdl)
plot(mdl$fitted.values, m.res,
     ylab="Residuals", xlab="Fitted Values of Gross",
     main="Movies - Fitted Gross Vs Residuals")
abline(0, 0)
```



There definitely exists a correlation as is evident in the residual plot above. It does not look randmly scattered as we would expect in a fully linearly fit model. It appears **biased** and **heteroscedastic**. To examine where the relations might be, let us do pairwise plot

```
df_temp = subset(train_data_numeric_nonNA_subset, train_data_numeric_nonNA_subset$Gross != 0)
suppressWarnings(ggpairs(df_temp)) +
  ggtitle("GGPairs plot for feature examination")
```




```
# Model with Transformed Numeric Variables
# ========================================
# Build & evaluate model 2 (transformed numeric variables only)

# feature transformation functions
# split a feature at mean value to bin into high or low (1 or 0)
feature_xform_to_binary = function(df, feature){
    feature_mean_val = mean(df[, feature])
    new_label = paste0(feature, '_bin')
    # assign all to 0
    df[new_label] = 0
    df[df[, feature]>feature_mean_val, new_label] = 1
```

```
    return(df)
}
```

Based on examining the pairwise plots we choose the variables to transform.

```
feature_transform = function(df){
  df = na.omit(df)
  df = feature_xform_to_binary(df, 'imdbVotes') #imdbVotes_bin - binary xform
  df = feature_xform_to_binary(df, 'Budget') #Budget_bin - binary xform

  df['Runtime_log'] = log(df$Runtime) # log xform

  df['imdbRating_P2'] = (df$imdbRating)^2 # power transform
  df['imdbVotes_P2'] = (df$imdbVotes)^2 # power transform
  df['imdbUserRating_P2'] = (df$tomatoUserRating)^2 # power transform
  return(df)
}

train_data_xform = feature_transform(train_data_numeric_nonNA_subset)
test_data_xform = feature_transform(train_data_numeric_nonNA_subset)

# call function to compute average training and test rsmes for range of sizes
resp_var = 'Gross'
fx = paste(resp_var, ' ~ ', '.', ' - ', resp_var)

# Fit a model and print results
xformModel<-lm(fx, data = train_data_xform)
print(summary(xformModel))
```

```
##
## Call:
## lm(formula = fx, data = train_data_xform)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -490280008  -32664987   -2942570   25832101 1297351733
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.993e+09  4.412e+08   4.518 6.46e-06 ***
## Year              -1.916e+06  4.042e+06  -0.474 0.635544
## Runtime            1.149e+06  4.356e+05   2.638 0.008384 **
## imdbRating         1.064e+08  1.733e+07   6.137 9.41e-10 ***
## imdbVotes          6.677e+02  4.414e+01  15.127  < 2e-16 ***
## tomatoMeter        6.967e+05  2.578e+05   2.703 0.006915 **
## tomatoRating      -6.863e+06  4.947e+06  -1.387 0.165395
## tomatoReviews     -2.138e+05  7.400e+04  -2.889 0.003885 **
## tomatoFresh        3.058e+05  1.016e+05   3.011 0.002624 **
## tomatoUserMeter    3.784e+05  2.559e+05   1.479 0.139229
## tomatoUserRating  -4.119e+07  5.553e+07  -0.742 0.458308
## tomatoUserReviews  5.253e+00  5.399e-01   9.729  < 2e-16 ***
## Budget             2.678e+00  6.817e-02  39.290  < 2e-16 ***
## Date               1.866e+06  3.955e+06   0.472 0.637140
## Decade            -7.363e+05  5.951e+05  -1.237 0.216126
```

```
## imdbVotes_bin     -4.511e+06  6.586e+06  -0.685 0.493505
## Budget_bin        -3.931e+07  5.374e+06  -7.316 3.20e-13 ***
## Runtime_log       -1.719e+08  5.079e+07  -3.386 0.000718 ***
## imdbRating_P2     -1.235e+07  1.536e+06  -8.044 1.20e-15 ***
## imdbVotes_P2      -2.003e-04  3.632e-05  -5.516 3.74e-08 ***
## imdbUserRating_P2  1.604e+07  8.217e+06   1.952 0.050995 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 96020000 on 3256 degrees of freedom
## Multiple R-squared:  0.6992, Adjusted R-squared:  0.6973
## F-statistic: 378.4 on 20 and 3256 DF,  p-value: < 2.2e-16
```

```
xformModel.rmse <- sqrt(mean(xformModel$residuals^2))
cat("Train RMSE is:", xformModel.rmse, "\n")
```

```
## Train RMSE is: 95707770
```

```
# Cross validation results
xformModel_cv <- cvFit(xformModel, data = train_data_xform,
                       y = train_data_xform$Gross,
                       K=10)
print(xformModel_cv)
```

```
## 10-fold CV results:
##       CV
## 97584745
```

Categorcial Variables

We have to convert genre, actors, directors, and other categorical variables to columns that can be used for regression (e.g. binary columns). To achieve that, we have to process variables such as awards into more useful columns.

```
# Model of Categorical Variables
# ==============================
# so that we use only newly converted columns, let us get back to our train_data
# Let us also ensure that the dataset is free from NA values
train_data_cat_var = train_data
```

Create helper functions for converting categorical variables into columns used for regression

```
commasplit_tokenizer = function(x)
    unlist(strsplit(as.character(x), ","))

# convert each row with binaries for categorical variables
binary_conversion = function(row_v, dict) {
    # Remove commas after feature elements using spl tokenizer
    gc = VCorpus(VectorSource(commasplit_tokenizer(row_v)))
    # wordlength is specified so that two letter countries like UK will be tokenized
    cnt = as.matrix(TermDocumentMatrix(gc, control = list(tokenize = commasplit_tokenizer,
                                                          dictionary=dict,
                                                          wordLengths=c(1,Inf),
                                                          tolower=FALSE)))
    rslt_list = suppressWarnings(as.numeric(qdapTools::counts2list(cnt)))
    rslt_list[is.na(rslt_list)] = 0
```

```
    # "t" transpose it as we use TDM (TermDocumentMatrix). If it were DTM, transpose is not needed
    return(t(rslt_list)[1,])
}

# convert a single categorical variable to columns
convert_cat_var = function(df, feature){
    # Get a dictionary of all possible feature elements (eg., list of all genres)
    feature_dict = sort(unique(unlist(tokenize_regex(df[,feature], pattern=", "))))
    # create binary columns of the feature in list form
    feature_conv = t(sapply(df[, feature], binary_conversion, dict=feature_dict))
    # change rownames as index
    rownames(feature_conv) = c(1:dim(feature_conv)[1])
    # change colnames to dict
    colnames(feature_conv) = feature_dict
    return(feature_conv)
}

# get the common top 20 in a category
get_top20_in_category = function(tr_df, tst_df, feature){
    # convert train feature factors into binary variables
    df_feature_conv_train = convert_cat_var(tr_df, feature)
    # convert test feature factors into binary variables
    df_feature_conv_test = convert_cat_var(tst_df, feature)

    # get the common top 20 of the selected feature (eg., genre) from both sets
    feature_freq_sorted_train = sort(colSums(df_feature_conv_train), decreasing = TRUE)
    common_features = intersect(names(feature_freq_sorted_train),
                                colnames(df_feature_conv_test))
    # get top 20 from the common set
    top20_feature_set = sort(common_features[1:20])

    df_top20_train = data.frame(df_feature_conv_train[,top20_feature_set])
    df_top20_train['index'] = c(1:dim(df_feature_conv_train)[1])

    df_top20_test = data.frame(df_feature_conv_test[,top20_feature_set])
    df_top20_test['index'] = c(1:dim(df_feature_conv_test)[1])
    return(list(df_top20_train, df_top20_test))
}

# main starter function to convert categories to columns
convert_cats_to_cols = function(tr_df, tst_df, features){
    rslt_top20_category = sapply(features, get_top20_in_category, tr_df=tr_df,
                                 tst_df=tst_df)
    top20_train_data = rslt_top20_category[1,]
    top20_test_data = rslt_top20_category[2,]
    df_encode_train = Reduce(function(df1, df2) {
        m <- merge(df1, df2, by='index', all=TRUE)
    }, top20_train_data)
    df_encode_test = Reduce(function(df1, df2) {
        m <- merge(df1, df2, by='index', all=TRUE)
    }, top20_test_data)
    return(list(df_encode_train, df_encode_test))
}
```

First, we have to pick the categorical variables to be used in this model. Of the non-numeric variables, we choose the more commonly known ones; genre, actors, director, language, country and Production. In addition, to reduce the number of features so that we don't overfit the model, we choose only the top 20 of each of the variables to encode.

```
enc_features = c('Genre', 'Director', 'Actors', 'Language', 'Country', 'Production')
rslt_data_enc = convert_cats_to_cols(na.omit(train_data), na.omit(test_data), enc_features)
train_data_cat_enc = rslt_data_enc[[1]]
test_data_cat_enc = rslt_data_enc[[2]]
```

Process Awards into useful columns.

```
compute_wins_nominations = function(s){
    # Extract win and nomination from Awards variable information using
    # regex serach patterns which are
    patterns = c("(\\d+)\\s.*?w[a-z]n.*\\s(\\d+).*?nominat",
                 "(\\d+)\\s.*?w[a-z]n",
                 "w[a-z]n.*\\s(\\d+).*?",
                 "(\\d+)\\s.*?nominat",
                 "nominat.*\\s(\\d+).*?")
    str_split = unlist(strsplit(s, "\\."))
    # Initialize wins and nominations
    wins = 0
    noms = 0
    # Loop through the string parts and compare with all patterns
    for (i in seq(1:length(str_split))){
        for (j in seq(1:length(patterns))){
            rslt = str_match(tolower(str_split[i]), patterns[j])
            if (!is.na(rslt[1])){
                rslt = suppressWarnings(as.numeric(rslt))
                if (j == 1){
                    # Both win and nomination
                    wins = wins + rslt[2]
                    noms = noms + rslt[3]
                    break
                }
                else if (j == 2 | j == 3){
                    # only win
                    wins = wins + rslt[2]
                    break
                }
                else if (j == 4 | j == 5){
                    # only nomination
                    noms = noms + rslt[2]
                    break
                }
            }
        }
    }
    return(c(wins, noms))
}

process_awards = function(df) {
    # get win and nomination by extracting awards
    awards_data = t(sapply(df$Awards, compute_wins_nominations))
```

```
    rownames(df) = c(1:dim(df)[1])
    rownames(awards_data) = c(1:dim(awards_data)[1])
    # set dataframe columns as wins and nominations
    colnames(awards_data) = c("wins", "nominations")
    df_awards_data = data.frame(awards_data)
    df_awards_data['index'] = c(1:dim(df_awards_data)[1])
    return(df_awards_data)
}

train_data_awards = process_awards(train_data)
test_data_awards = process_awards(test_data)
```

Merge the categories data and the awards data into a dataset for train and test

```
train_data_cat_awards = merge(train_data_cat_enc, train_data_awards,
                                    by='index', all=TRUE)
test_data_cat_awards = merge(test_data_cat_enc, test_data_awards,
                                   by='index', all=TRUE)
```

Now that we have the merged datasets, let us slightly modify the datasets and we shall train the model to predict Gross as we did before, but with a different set of variables than the numeric variables

```
# Add the numeric response variable "Gross" and remove "index" column
train_data_cat_awards['Gross'] = train_data['Gross']
test_data_cat_awards['Gross'] = test_data['Gross']
```

Perform linear regression with this data

```
# call function to compute average training and test rsmes for range of sizes
resp_var = 'Gross'
fx = paste(resp_var, ' ~ ', '.', ' - ', resp_var)

# Fit a model and print results
catModel<-lm(fx, data = na.omit(train_data_cat_awards))
print(summary(catModel))
```

```
##
## Call:
## lm(formula = fx, data = na.omit(train_data_cat_awards))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -603160220  -63036299  -36671672   17316257 2309909262
##
## Coefficients: (4 not defined because of singularities)
##                                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)                           143094118   89589757   1.597  0.11032
## index                                       -173       3099  -0.056  0.95549
## Action                                -165235168   71598601  -2.308  0.02108
## Adventure                             -173125020   72239209  -2.397  0.01661
## Animation                             -156417019   72616729  -2.154  0.03132
## Biography                             -157975989   72404269  -2.182  0.02920
## Comedy                                -161910833   71561201  -2.263  0.02373
## Crime                                 -149665249   72269942  -2.071  0.03845
## Documentary                           -173790779   74017672  -2.348  0.01894
## Drama                                 -162757504   71614822  -2.273  0.02311
```

```
## Family                  -130774975   90830852  -1.440  0.15004
## Fantasy                  -93020438   76932877  -1.209  0.22671
## Film.Noir                 97232621  186104112   0.522  0.60139
## History                         NA         NA      NA       NA
## Horror                  -155754462   72553338  -2.147  0.03189
## Music                           NA         NA      NA       NA
## Musical                 -331445409  134579226  -2.463  0.01384
## Mystery                 -213169434   81189830  -2.626  0.00869
## Romance                 -191728619   93145518  -2.058  0.03964
## Sci.Fi                  -105805842   83742072  -1.263  0.20652
## Thriller                -116747210   88860202  -1.314  0.18900
## Western                         NA         NA      NA       NA
## Alfred.Hitchcock         -23002925   61992468  -0.371  0.71062
## Barry.Levinson            -3328815   53438632  -0.062  0.95033
## Brian.De.Palma            26753476   57965106   0.462  0.64444
## Chris.Columbus            35484310   50577261   0.702  0.48299
## Clint.Eastwood            21296417   45499926   0.468  0.63978
## David.Fincher            -51430224   53453382  -0.962  0.33605
## Francis.Ford.Coppola     -51780324   51394483  -1.008  0.31377
## Martin.Scorsese           19350993   42344187   0.457  0.64771
## Oliver.Stone             -44552903   48698204  -0.915  0.36033
## Renny.Harlin              -9221170   43025176  -0.214  0.83031
## Richard.Donner           -61437288   51006248  -1.205  0.22849
## Ridley.Scott             -19524721   41300476  -0.473  0.63643
## Robert.Zemeckis           10355160   51218129   0.202  0.83979
## Ron.Howard               -25807212   52345483  -0.493  0.62204
## Sam.Raimi                  5236344   54308246   0.096  0.92319
## Spike.Lee                -48673041   48600954  -1.001  0.31667
## Steven.Soderbergh        -48611684   46915027  -1.036  0.30021
## Steven.Spielberg         -32548696   38094923  -0.854  0.39294
## Tim.Burton               -17904278   48411052  -0.370  0.71153
## Woody.Allen               30999185   40266120   0.770  0.44144
## Adam.Sandler              27861636   36714977   0.759  0.44799
## Al.Pacino                 37028530   45376118   0.816  0.41454
## Arnold.Schwarzenegger     27881601   40037908   0.696  0.48624
## Ben.Stiller               26638219   46090490   0.578  0.56334
## Brad.Pitt                 -9411752   41685254  -0.226  0.82139
## Bruce.Willis             -44988158   35950378  -1.251  0.21089
## Denzel.Washington         -7158765   41829280  -0.171  0.86412
## Eddie.Murphy               8022359   42592357   0.188  0.85061
## George.Clooney           -28235622   41588056  -0.679  0.49723
## John.Cusack              -15367353   44657879  -0.344  0.73078
## John.Travolta            -30709875   40871789  -0.751  0.45249
## Johnny.Depp                691838   35107646   0.020  0.98428
## Kevin.Costner             33533743   41571777   0.807  0.41993
## Nicolas.Cage              45068340   35999327   1.252  0.21069
## Robert.De.Niro            -7867996   32927627  -0.239  0.81116
## Sean.Connery             -29590429   41319234  -0.716  0.47396
## Sylvester.Stallone         9546315   37184576   0.257  0.79741
## Tom.Cruise                70063023   45301946   1.547  0.12207
## Tom.Hanks                  6888451   42002992   0.164  0.86974
## Will.Ferrell             -30516910   44395374  -0.687  0.49189
## Arabic                   618200126  122369940   5.052 4.63e-07
## Cantonese                  2577256   87985126   0.029  0.97663
```

```
## Dari                                          -25556382  164868098  -0.155  0.87682
## Dutch                                         -31073774  200568296  -0.155  0.87689
## English                                        54215689   46688408   1.161  0.24564
## Filipino                                      -10811220  124639597  -0.087  0.93088
## French                                         52742851   58515359   0.901  0.36747
## German                                        137243079  122172774   1.123  0.26138
## Greek                                          42952407  166913884   0.257  0.79694
## Hebrew                                         58653528  126924946   0.462  0.64403
## Hindi                                          72770160  100568375   0.724  0.46937
## Italian                                       121543978  189247225   0.642  0.52076
## Japanese                                       41881302  152920703   0.274  0.78420
## Korean                                         17156570  166864086   0.103  0.91811
## Mandarin                                       33048279  111724000   0.296  0.76740
## N.A                                             7859099  121085101   0.065  0.94825
## Norwegian                                      54320007  166971443   0.325  0.74496
## Portuguese                                     17592479  121747928   0.144  0.88512
## Spanish                                       -18844789   70493472  -0.267  0.78924
## Thai                                           51563322  124551937   0.414  0.67891
## Australia                                       2544348   41478356   0.061  0.95109
## Canada                                         53117599   37840975   1.404  0.16051
## China                                         -27678671   86685360  -0.319  0.74952
## Czech.Republic                                 5326638   117584902   0.045  0.96387
## Denmark                                        60150956   78079624   0.770  0.44113
## France                                         27864192   38303223   0.727  0.46700
## Germany                                        26955456   38499910   0.700  0.48389
## Hong.Kong                                      41443952   76106609   0.545  0.58610
## India                                           -293802   87078784  -0.003  0.99731
## Ireland                                        19537842   62346814   0.313  0.75402
## Israel                                               NA         NA      NA       NA
## Italy                                          53794980   98486554   0.546  0.58496
## Japan                                         -20862312   98803399  -0.211  0.83278
## Mexico                                         10310749   83876111   0.123  0.90217
## Netherlands                                   -29544126  161473125  -0.183  0.85484
## New.Zealand                                    16321930   58047752   0.281  0.77859
## South.Africa                                   20928786   72675269   0.288  0.77338
## Spain                                          47083427   49119456   0.959  0.33786
## UK                                             42456131   34073618   1.246  0.21286
## USA                                            23239158   32971805   0.705  0.48098
## X20th.Century.Fox                             -19199699   12254718  -1.567  0.11728
## Buena.Vista.Pictures                          -30245454   25380784  -1.192  0.23348
## Columbia.Pictures                              5364227   18398796   0.292  0.77065
## Focus.Features                                 10061595   23537107   0.427  0.66906
## Lionsgate                                      76376723   27783000   2.749  0.00601
## Lionsgate.Films                                -8746931   26931928  -0.325  0.74537
## MGM                                           -33953409   24091869  -1.409  0.15884
## Miramax.Films                                  11781449   20964900   0.562  0.57418
## New.Line.Cinema                                18264580   18893870   0.967  0.33377
## Paramount.Pictures                              8134889   12808458   0.635  0.52540
## Sony.Pictures                                   -498750   17098402  -0.029  0.97673
## Sony.Pictures.Classics                        -21592371   20007940  -1.079  0.28059
## Sony.Pictures.Home.Entertainment               -1385586   19551515  -0.071  0.94351
## The.Weinstein.Company                          19630834   27527129   0.713  0.47581
## Twentieth.Century.Fox.Home.Entertainment       -1091019   26241984  -0.042  0.96684
## Universal.Pictures                             -1756426   12399420  -0.142  0.88736
```

```
## Walt.Disney.Pictures                  -20351934   22545511  -0.903  0.36675
## Warner.Bros.                           -2685116   25674803  -0.105  0.91671
## Warner.Bros..Pictures                 -10351803   11949789  -0.866  0.38641
## Warner.Home.Video                     -22464952   23384992  -0.961  0.33680
## wins                                     127211     334331   0.380  0.70360
## nominations                              2621340     215452  12.167  < 2e-16
##
## (Intercept)
## index
## Action                                *
## Adventure                             *
## Animation                             *
## Biography                             *
## Comedy                                *
## Crime                                 *
## Documentary                           *
## Drama                                 *
## Family
## Fantasy
## Film.Noir
## History
## Horror                                *
## Music
## Musical                               *
## Mystery                               **
## Romance                               *
## Sci.Fi
## Thriller
## Western
## Alfred.Hitchcock
## Barry.Levinson
## Brian.De.Palma
## Chris.Columbus
## Clint.Eastwood
## David.Fincher
## Francis.Ford.Coppola
## Martin.Scorsese
## Oliver.Stone
## Renny.Harlin
## Richard.Donner
## Ridley.Scott
## Robert.Zemeckis
## Ron.Howard
## Sam.Raimi
## Spike.Lee
## Steven.Soderbergh
## Steven.Spielberg
## Tim.Burton
## Woody.Allen
## Adam.Sandler
## Al.Pacino
## Arnold.Schwarzenegger
## Ben.Stiller
## Brad.Pitt
```

```
## Bruce.Willis
## Denzel.Washington
## Eddie.Murphy
## George.Clooney
## John.Cusack
## John.Travolta
## Johnny.Depp
## Kevin.Costner
## Nicolas.Cage
## Robert.De.Niro
## Sean.Connery
## Sylvester.Stallone
## Tom.Cruise
## Tom.Hanks
## Will.Ferrell
## Arabic                                    ***
## Cantonese
## Dari
## Dutch
## English
## Filipino
## French
## German
## Greek
## Hebrew
## Hindi
## Italian
## Japanese
## Korean
## Mandarin
## N.A
## Norwegian
## Portuguese
## Spanish
## Thai
## Australia
## Canada
## China
## Czech.Republic
## Denmark
## France
## Germany
## Hong.Kong
## India
## Ireland
## Israel
## Italy
## Japan
## Mexico
## Netherlands
## New.Zealand
## South.Africa
## Spain
## UK
```

```
## USA
## X20th.Century.Fox
## Buena.Vista.Pictures
## Columbia.Pictures
## Focus.Features
## Lionsgate                                  **
## Lionsgate.Films
## MGM
## Miramax.Films
## New.Line.Cinema
## Paramount.Pictures
## Sony.Pictures
## Sony.Pictures.Classics
## Sony.Pictures.Home.Entertainment
## The.Weinstein.Company
## Twentieth.Century.Fox.Home.Entertainment
## Universal.Pictures
## Walt.Disney.Pictures
## Warner.Bros.
## Warner.Bros..Pictures
## Warner.Home.Video
## wins
## nominations                                ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 157900000 on 3072 degrees of freedom
## Multiple R-squared:  0.1778, Adjusted R-squared:  0.1459
## F-statistic: 5.582 on 119 and 3072 DF,  p-value: < 2.2e-16
```

```
catModel.rmse <- sqrt(mean(catModel$residuals^2))
cat("Train RMSE of the category variable model is:", catModel.rmse, "\n")
```

```
## Train RMSE of the category variable model is: 154925828
```

```
# Cross validation results
catModel_cv <- cvFit(catModel, data = train_data_cat_awards,
                        y = train_data_cat_awards$Gross,
                        K=10)
```

```
## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
```

```
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(...): prediction from a rank-deficient fit may be
## misleading
```

```
print(catModel_cv)
```

```
## 10-fold CV results:
## CV
## NA
```

Interactions

```
intModel <- lm(Gross ~ . + .:., data = train_data_numeric_nonNA_subset)
print(summary(intModel))
```

```
##
## Call:
## lm(formula = Gross ~ . + .:., data = train_data_numeric_nonNA_subset)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -465031638  -27233521   -3572198   17496520  926797483
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     4.668e+10  4.112e+10   1.135 0.256425
## Year                            2.271e+08  1.003e+09   0.226 0.820953
## Runtime                         5.644e+07  1.292e+07   4.369 1.29e-05 ***
## imdbRating                      8.812e+08  8.441e+08   1.044 0.296578
## imdbVotes                       4.737e+03  4.830e+03   0.981 0.326815
## tomatoMeter                    -4.179e+07  3.663e+07  -1.141 0.254054
## tomatoRating                   -2.124e+07  7.352e+08  -0.029 0.976958
## tomatoReviews                  -4.274e+07  2.252e+07  -1.898 0.057805 .
## tomatoFresh                     9.964e+07  3.219e+07   3.095 0.001982 **
## tomatoUserMeter                 3.700e+07  5.101e+07   0.725 0.468232
## tomatoUserRating               -4.765e+09  2.043e+09  -2.333 0.019730 *
## tomatoUserReviews               1.349e+03  3.669e+02   3.676 0.000241 ***
## Budget                          2.013e+01  2.361e+01   0.853 0.393902
## Date                           -3.684e+08  1.005e+09  -0.367 0.713981
## Decade                          9.711e+07  1.201e+08   0.809 0.418643
## Year:Runtime                   -3.083e+05  1.428e+05  -2.159 0.030899 *
## Year:imdbRating                -3.712e+06  7.279e+06  -0.510 0.610139
## Year:imdbVotes                  1.168e+02  7.489e+01   1.559 0.119056
## Year:tomatoMeter                6.087e+05  3.958e+05   1.538 0.124202
```

```
## Year:tomatoRating                -4.552e+06  8.562e+06  -0.532 0.595039
## Year:tomatoReviews                2.300e+05  2.164e+05   1.063 0.288073
## Year:tomatoFresh                 -3.280e+05  3.068e+05  -1.069 0.285151
## Year:tomatoUserMeter             -1.528e+04  5.984e+05  -0.026 0.979631
## Year:tomatoUserRating             3.799e+06  2.317e+07   0.164 0.869812
## Year:tomatoUserReviews           -7.463e+01  5.411e+01  -1.379 0.167972
## Year:Budget                       5.411e-01  2.424e-01   2.233 0.025635 *
## Year:Date                         5.587e+04  6.021e+04   0.928 0.353508
## Year:Decade                      -1.549e+05  5.097e+05  -0.304 0.761177
## Runtime:imdbRating               -4.559e+05  2.227e+05  -2.047 0.040703 *
## Runtime:imdbVotes                 3.172e+00  7.110e-01   4.462 8.41e-06 ***
## Runtime:tomatoMeter               1.398e+04  1.234e+04   1.133 0.257380
## Runtime:tomatoRating             -2.094e+05  2.378e+05  -0.880 0.378703
## Runtime:tomatoReviews             4.579e+03  3.811e+03   1.201 0.229696
## Runtime:tomatoFresh              -3.420e+03  4.887e+03  -0.700 0.484091
## Runtime:tomatoUserMeter           2.958e+04  1.266e+04   2.336 0.019559 *
## Runtime:tomatoUserRating         -8.654e+05  4.663e+05  -1.856 0.063529 .
## Runtime:tomatoUserReviews         4.943e-02  2.936e-02   1.683 0.092428 .
## Runtime:Budget                   -8.710e-03  2.492e-03  -3.494 0.000482 ***
## Runtime:Date                      2.348e+05  1.364e+05   1.722 0.085235 .
## Runtime:Decade                    4.731e+04  2.912e+04   1.625 0.104315
## imdbRating:imdbVotes             -3.117e+02  5.094e+01  -6.119 1.06e-09 ***
## imdbRating:tomatoMeter            3.953e+05  3.912e+05   1.011 0.312331
## imdbRating:tomatoRating          -8.857e+06  6.680e+06  -1.326 0.184949
## imdbRating:tomatoReviews         -2.879e+05  1.261e+05  -2.283 0.022514 *
## imdbRating:tomatoFresh            2.517e+05  2.213e+05   1.138 0.255365
## imdbRating:tomatoUserMeter       -9.024e+05  3.392e+05  -2.660 0.007853 **
## imdbRating:tomatoUserRating       1.998e+07  1.297e+07   1.540 0.123578
## imdbRating:tomatoUserReviews     -5.843e+00  2.096e+00  -2.787 0.005349 **
## imdbRating:Budget                -2.496e-01  1.448e-01  -1.724 0.084766 .
## imdbRating:Date                   3.397e+06  7.214e+06   0.471 0.637768
## imdbRating:Decade                -9.255e+04  1.122e+06  -0.082 0.934258
## imdbVotes:tomatoMeter             4.904e-01  3.399e+00   0.144 0.885293
## imdbVotes:tomatoRating            6.987e+01  4.755e+01   1.469 0.141804
## imdbVotes:tomatoReviews          -2.163e+00  9.520e-01  -2.272 0.023142 *
## imdbVotes:tomatoFresh             6.742e-01  1.073e+00   0.628 0.529795
## imdbVotes:tomatoUserMeter         6.683e+00  3.516e+00   1.901 0.057384 .
## imdbVotes:tomatoUserRating       -5.169e+02  9.371e+01  -5.516 3.76e-08 ***
## imdbVotes:tomatoUserReviews      -7.699e-06  3.765e-06  -2.045 0.040950 *
## imdbVotes:Budget                  3.977e-06  3.762e-07  10.571  < 2e-16 ***
## imdbVotes:Date                   -1.308e+02  7.499e+01  -1.744 0.081286 .
## imdbVotes:Decade                  1.340e+01  5.213e+00   2.571 0.010178 *
## tomatoMeter:tomatoRating          1.796e+05  1.322e+05   1.358 0.174499
## tomatoMeter:tomatoReviews        -1.134e+05  4.099e+04  -2.768 0.005680 **
## tomatoMeter:tomatoFresh           7.892e+03  1.108e+04   0.712 0.476286
## tomatoMeter:tomatoUserMeter      -4.317e+04  2.992e+04  -1.443 0.149213
## tomatoMeter:tomatoUserRating      5.300e+05  1.149e+06   0.461 0.644700
## tomatoMeter:tomatoUserReviews    -7.507e-01  1.561e-01  -4.808 1.59e-06 ***
## tomatoMeter:Budget                6.640e-03  9.339e-03   0.711 0.477148
## tomatoMeter:Date                 -5.004e+05  3.868e+05  -1.294 0.195852
## tomatoMeter:Decade               -8.933e+04  8.137e+04  -1.098 0.272339
## tomatoRating:tomatoReviews        1.788e+05  1.707e+05   1.047 0.295000
## tomatoRating:tomatoFresh         -3.565e+05  2.165e+05  -1.646 0.099770 .
## tomatoRating:tomatoUserMeter      1.131e+06  5.904e+05   1.916 0.055513 .
```

```
## tomatoRating:tomatoUserRating      -3.366e+07  2.212e+07  -1.522 0.128111
## tomatoRating:tomatoUserReviews      6.408e+00  2.124e+00   3.018 0.002567 **
## tomatoRating:Budget                 2.002e-01  1.832e-01   1.093 0.274511
## tomatoRating:Date                   2.541e+06  8.428e+06   0.301 0.763098
## tomatoRating:Decade                 2.079e+06  1.570e+06   1.324 0.185488
## tomatoReviews:tomatoFresh           3.303e+03  1.013e+03   3.260 0.001126 **
## tomatoReviews:tomatoUserMeter      -3.071e+03  9.941e+03  -0.309 0.757407
## tomatoReviews:tomatoUserRating      7.843e+05  4.040e+05   1.942 0.052274 .
## tomatoReviews:tomatoUserReviews    -2.042e-01  4.600e-02  -4.439 9.35e-06 ***
## tomatoReviews:Budget               -1.701e-04  1.757e-03  -0.097 0.922863
## tomatoReviews:Date                 -2.275e+05  2.150e+05  -1.058 0.290100
## tomatoReviews:Decade                1.816e+04  2.251e+04   0.807 0.419704
## tomatoFresh:tomatoUserMeter        -4.784e+03  1.530e+04  -0.313 0.754576
## tomatoFresh:tomatoUserRating        3.108e+05  5.583e+05   0.557 0.577803
## tomatoFresh:tomatoUserReviews       3.460e-01  4.861e-02   7.118 1.34e-12 ***
## tomatoFresh:Budget                 -4.740e-03  2.492e-03  -1.902 0.057279 .
## tomatoFresh:Date                    3.104e+05  3.046e+05   1.019 0.308383
## tomatoFresh:Decade                 -2.742e+04  3.212e+04  -0.854 0.393315
## tomatoUserMeter:tomatoUserRating    1.109e+06  3.993e+05   2.778 0.005501 **
## tomatoUserMeter:tomatoUserReviews   2.935e-02  1.132e-01   0.259 0.795454
## tomatoUserMeter:Budget             -4.092e-02  9.335e-03  -4.383 1.21e-05 ***
## tomatoUserMeter:Date               -7.169e+04  5.852e+05  -0.123 0.902501
## tomatoUserMeter:Decade              6.681e+04  7.271e+04   0.919 0.358237
## tomatoUserRating:tomatoUserReviews  7.381e+00  3.360e+00   2.196 0.028137 *
## tomatoUserRating:Budget             3.226e+00  3.387e-01   9.525  < 2e-16 ***
## tomatoUserRating:Date               4.685e+06  2.272e+07   0.206 0.836661
## tomatoUserRating:Decade            -6.137e+06  2.954e+06  -2.077 0.037841 *
## tomatoUserReviews:Budget            3.528e-08  1.650e-08   2.139 0.032533 *
## tomatoUserReviews:Date              7.437e+01  5.411e+01   1.375 0.169383
## tomatoUserReviews:Decade           -4.110e-01  2.708e-01  -1.518 0.129220
## Budget:Date                        -4.592e-01  2.400e-01  -1.913 0.055840 .
## Budget:Decade                      -9.503e-02  1.658e-02  -5.731 1.09e-08 ***
## Date:Decade                         1.095e+05  5.035e+05   0.217 0.827895
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 79230000 on 3171 degrees of freedom
## Multiple R-squared:  0.8005, Adjusted R-squared:  0.7939
## F-statistic: 121.2 on 105 and 3171 DF,  p-value: < 2.2e-16
```

```
intModel.rmse <- sqrt(mean(intModel$residuals^2))
cat("Train RMSE for Interaction model is:", intModel.rmse, "\n")
```

```
## Train RMSE for Interaction model is: 77935567
```

```
# Cross validation results
intModel_cv <- cvFit(intModel, data = train_data_numeric_nonNA_subset,
                        y = train_data_numeric_nonNA_subset$Gross,
                        K=10)
print(intModel_cv)
```

```
## 10-fold CV results:
##       CV
## 89203508
```

Final model

transforms + interactions added to the existing numeric variables

```
finalModel <- lm(Gross ~ . +
                   Runtime:imdbVotes +
                   Runtime:tomatoUserReviews +
                   imdbRating:imdbVotes +
                   imdbRating:tomatoUserReviews +
                   imdbRating:Budget +
                   imdbVotes:tomatoUserRating +
                   imdbVotes:Budget +
                   tomatoMeter:tomatoReviews +
                   tomatoMeter:tomatoUserReviews +
                   tomatoReviews:tomatoUserReviews +
                   tomatoFresh:tomatoUserReviews +
                   tomatoUserMeter:Budget +
                   tomatoUserRating:tomatoUserReviews +
                   tomatoUserRating:Budget,
                   data = train_data_xform)
print(summary(finalModel))
```

```
##
## Call:
## lm(formula = Gross ~ . + Runtime:imdbVotes + Runtime:tomatoUserReviews +
##     imdbRating:imdbVotes + imdbRating:tomatoUserReviews + imdbRating:Budget +
##     imdbVotes:tomatoUserRating + imdbVotes:Budget + tomatoMeter:tomatoReviews +
##     tomatoMeter:tomatoUserReviews + tomatoReviews:tomatoUserReviews +
##     tomatoFresh:tomatoUserReviews + tomatoUserMeter:Budget +
##     tomatoUserRating:tomatoUserReviews + tomatoUserRating:Budget,
##     data = train_data_xform)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -524207016  -28591517   -4972142   16181641  952899272
##
## Coefficients:
##                                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       1.485e+09  3.931e+08   3.779 0.000161 ***
## Year                              4.679e+06  3.542e+06   1.321 0.186687
## Runtime                           3.261e+05  3.999e+05   0.816 0.414829
## imdbRating                        4.941e+07  1.609e+07   3.070 0.002158 **
## imdbVotes                         2.838e+03  2.190e+02  12.958  < 2e-16 ***
## tomatoMeter                      -9.097e+04  2.324e+05  -0.392 0.695429
## tomatoRating                      1.256e+07  4.415e+06   2.844 0.004481 **
## tomatoReviews                    -1.135e+05  7.240e+04  -1.567 0.117158
## tomatoFresh                       1.286e+07  4.209e+06   3.055 0.002270 **
## tomatoUserMeter                   6.084e+05  2.884e+05   2.110 0.034969 *
## tomatoUserRating                 -1.447e+08  5.078e+07  -2.850 0.004394 **
## tomatoUserReviews                 2.454e+01  7.749e+00   3.167 0.001554 **
## Budget                           -4.471e+00  6.159e-01  -7.258 4.88e-13 ***
## Date                             -4.653e+06  3.465e+06  -1.343 0.179449
## Decade                           -5.476e+05  5.187e+05  -1.056 0.291233
## imdbVotes_bin                    -7.096e+06  5.921e+06  -1.198 0.230823
## Budget_bin                        8.674e+06  5.144e+06   1.686 0.091873 .
```

```
## Runtime_log                          -8.863e+07  4.510e+07  -1.965 0.049503 *
## imdbRating_P2                        -5.357e+06  1.441e+06  -3.719 0.000204 ***
## imdbVotes_P2                         -8.669e-05  5.258e-05  -1.649 0.099259 .
## imdbUserRating_P2                     1.962e+07  7.628e+06   2.571 0.010174 *
## Runtime:imdbVotes                     1.142e+00  5.123e-01   2.229 0.025874 *
## Runtime:tomatoUserReviews             1.346e-01  1.994e-02   6.750 1.74e-11 ***
## imdbRating:imdbVotes                 -1.933e+02  3.555e+01  -5.438 5.80e-08 ***
## imdbRating:tomatoUserReviews         -1.029e+01  1.202e+00  -8.557  < 2e-16 ***
## imdbRating:Budget                    -4.388e-01  1.054e-01  -4.163 3.23e-05 ***
## imdbVotes:tomatoUserRating           -2.637e+02  6.277e+01  -4.200 2.74e-05 ***
## imdbVotes:Budget                      2.222e-06  2.594e-07   8.568  < 2e-16 ***
## tomatoMeter:tomatoReviews            -1.291e+05  4.205e+04  -3.070 0.002158 **
## tomatoMeter:tomatoUserReviews         1.843e-02  6.844e-02   0.269 0.787731
## tomatoReviews:tomatoUserReviews      -1.601e-01  3.427e-02  -4.671 3.11e-06 ***
## tomatoFresh:tomatoUserReviews         2.000e-01  4.252e-02   4.704 2.66e-06 ***
## tomatoUserMeter:Budget               -9.630e-03  6.573e-03  -1.465 0.142999
## tomatoUserRating:tomatoUserReviews    1.210e+01  2.128e+00   5.689 1.39e-08 ***
## tomatoUserRating:Budget               2.751e+00  2.415e-01  11.390  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 83540000 on 3242 degrees of freedom
## Multiple R-squared:  0.7732, Adjusted R-squared:  0.7709
## F-statistic: 325.1 on 34 and 3242 DF,  p-value: < 2.2e-16
```

```
finalModel.rmse <- sqrt(mean(finalModel$residuals^2))
cat("Train RMSE for Final model is:", finalModel.rmse, "\n")
```

```
## Train RMSE for Final model is: 83096040
```

```
# Cross validation results
finalModel_cv <- cvFit(finalModel, data = train_data_xform,
                       y = train_data_numeric_nonNA_subset$Gross,
                       K=10)
print(finalModel_cv)
```

```
## 10-fold CV results:
##       CV
## 88082565
```

Lasso model

```
#train_temp <- subset(train_data_xform, select=-c(Gross))
baseline_log_linear<-lm(Gross~., data = train_data_xform)
print(summary(baseline_log_linear))
```

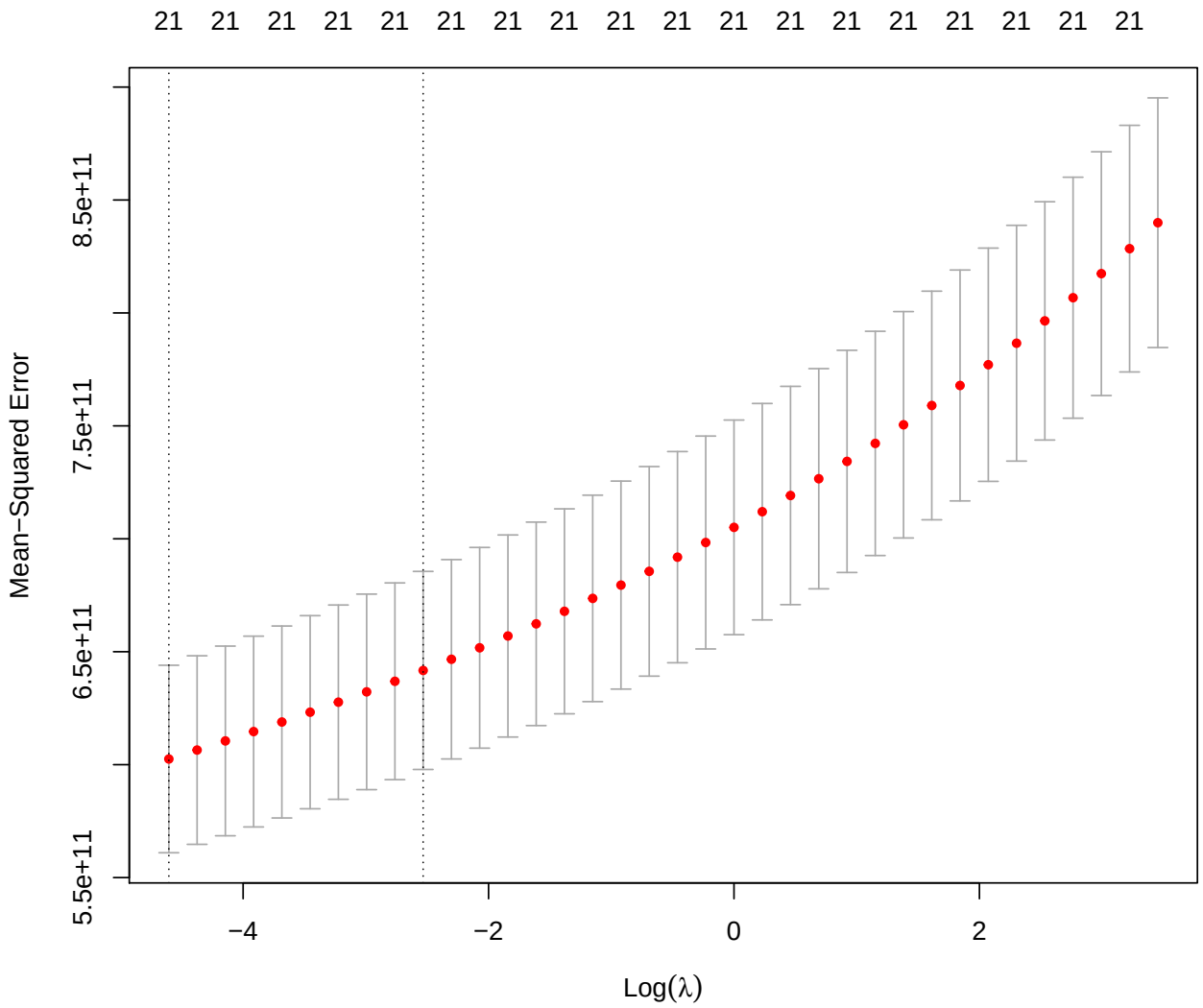
```
##
## Call:
## lm(formula = Gross ~ ., data = train_data_xform)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -490280008  -32664987   -2942570   25832101 1297351733
##
## Coefficients:
```

```
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.993e+09  4.412e+08   4.518 6.46e-06 ***
## Year              -1.916e+06  4.042e+06  -0.474 0.635544
## Runtime            1.149e+06  4.356e+05   2.638 0.008384 **
## imdbRating         1.064e+08  1.733e+07   6.137 9.41e-10 ***
## imdbVotes          6.677e+02  4.414e+01  15.127  < 2e-16 ***
## tomatoMeter        6.967e+05  2.578e+05   2.703 0.006915 **
## tomatoRating      -6.863e+06  4.947e+06  -1.387 0.165395
## tomatoReviews     -2.138e+05  7.400e+04  -2.889 0.003885 **
## tomatoFresh        3.058e+05  1.016e+05   3.011 0.002624 **
## tomatoUserMeter    3.784e+05  2.559e+05   1.479 0.139229
## tomatoUserRating  -4.119e+07  5.553e+07  -0.742 0.458308
## tomatoUserReviews  5.253e+00  5.399e-01   9.729  < 2e-16 ***
## Budget             2.678e+00  6.817e-02  39.290  < 2e-16 ***
## Date               1.866e+06  3.955e+06   0.472 0.637140
## Decade            -7.363e+05  5.951e+05  -1.237 0.216126
## imdbVotes_bin     -4.511e+06  6.586e+06  -0.685 0.493505
## Budget_bin        -3.931e+07  5.374e+06  -7.316 3.20e-13 ***
## Runtime_log       -1.719e+08  5.079e+07  -3.386 0.000718 ***
## imdbRating_P2     -1.235e+07  1.536e+06  -8.044 1.20e-15 ***
## imdbVotes_P2      -2.003e-04  3.632e-05  -5.516 3.74e-08 ***
## imdbUserRating_P2  1.604e+07  8.217e+06   1.952 0.050995 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 96020000 on 3256 degrees of freedom
## Multiple R-squared:  0.6992, Adjusted R-squared:  0.6973
## F-statistic: 378.4 on 20 and 3256 DF,  p-value: < 2.2e-16
```

```
# set lambda sequence to use for lasso
#lambdas = 10^seq(-10,1.5,0.1)
lambdas = 10^seq(-2,1.5,0.1)
#lasso
# train_Lasso <- train_data_numeric_nonNA_subset[,log_transform]
fm.lasso = glmnet(as.matrix(train_data_xform),
                  as.double(train_data_xform$Gross),
                  alpha = 1, lambda = lambdas, standardize = TRUE, thresh = 1e-12)
summary(fm.lasso)
```

```
##           Length Class     Mode
## a0         36    -none-    numeric
## beta      756    dgCMatrix S4
## df         36    -none-    numeric
## dim         2    -none-    numeric
## lambda     36    -none-    numeric
## dev.ratio  36    -none-    numeric
## nulldev     1    -none-    numeric
## npasses     1    -none-    numeric
## jerr        1    -none-    numeric
## offset      1    -none-    logical
## call        7    -none-    call
## nobs        1    -none-    numeric
```

```
cv_fit<-cv.glmnet(as.matrix(train_data_xform), train_data_xform$Gross, lambda = lambdas, standardize = T
plot(cv_fit)
```




```
opt_lambda<-cv_fit$lambda.min
print(opt_lambda)
```

```
## [1] 0.01
```

```
#summary(cv_fit)
#plot(fm.lasso, xvar ="lambda")
lasso.predict = predict(glmnet(as.matrix(train_data_xform), train_data_xform$Gross, alpha = 1, lambda =
sqrt(mean((lasso.predict - train_data_xform$Gross)^2))
```

```
## [1] 902590.4
```

```
# # Different approach to calculate Lasso RMSE
lasso_reg <-cv.glmnet(as.matrix(train_data_xform), train_data_xform$Gross, alpha = 1, lambda = lambdas,

lambda_best <- lasso_reg$lambda.min
lambda_best
```

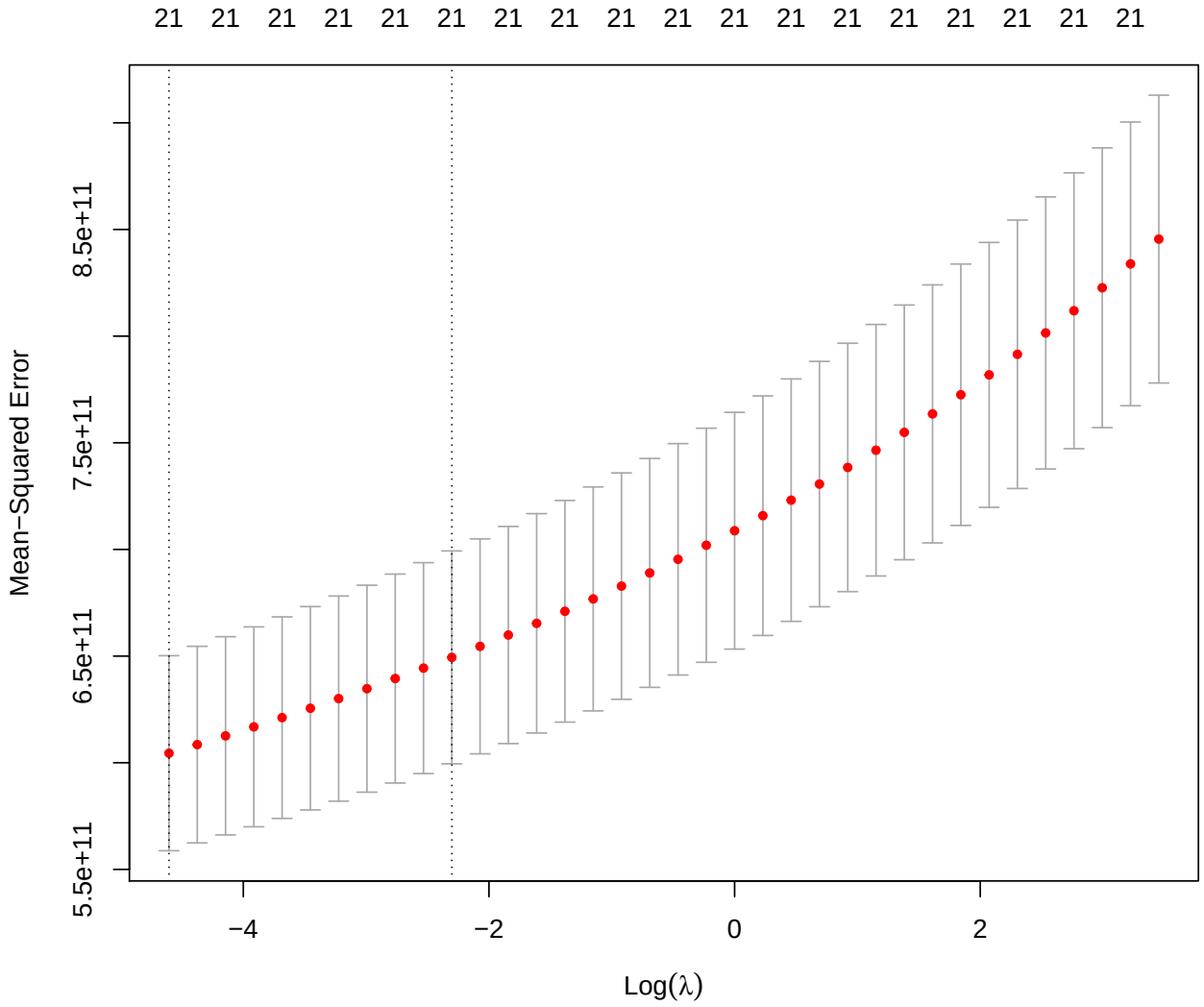
```
## [1] 0.01
```

```
lasso_model <- glmnet(as.matrix(train_data_xform), train_data_xform$Gross, alpha = 1, lambda = lambda_b

prediction_train <- predict(lasso_model, s = lambda_best, newx = as.matrix(train_data_xform))
```

```
sqrt(mean((prediction_train - train_data_xform$Gross)^2)) ##RMSE
```

```
## [1] 902590.4
```

```
fm.ridge = glmnet(as.matrix(train_data_xform), train_data_xform$Gross, alpha = 0, lambda = lambdas, sta
#summary(fm.ridge)
#plot(fm.ridge, xvar ="lambda")
cv_fit_ridge<-cv.glmnet(as.matrix(train_data_xform), train_data_xform$Gross, lambda = lambdas, nfolds =
plot(cv_fit_ridge)
```




```
opt_lambda<-cv_fit_ridge$lambda.min
print(opt_lambda)
```

```
## [1] 0.01
```

```
## -2.6521  -0.1649  -0.0155   0.0806   4.4767
##
## Coefficients: (1 not defined because of singularities)
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.290e+00  1.690e+01  -0.254 0.799627
## Year              -6.344e-01  1.628e-01  -3.898 9.69e-05 ***
## Runtime            1.599e-02  3.994e-03   4.003 6.24e-05 ***
## imdbVotes          5.023e-06  1.234e-06   4.071 4.69e-05 ***
## tomatoMeter       -1.827e-02  1.232e-02  -1.483 0.138137
## tomatoRating       1.623e+00  2.636e-01   6.155 7.50e-10 ***
## tomatoReviews      1.836e-02  4.667e-03   3.933 8.38e-05 ***
## tomatoFresh       -1.834e-02  6.132e-03  -2.990 0.002786 **
## tomatoRotten              NA         NA      NA       NA
## tomatoUserMeter    1.936e-01  1.356e-02  14.274  < 2e-16 ***
## tomatoUserRating   1.400e+00  4.126e-01   3.393 0.000692 ***
## tomatoUserReviews  6.655e-08  1.937e-08   3.435 0.000592 ***
## Budget            -3.562e-10  3.200e-09  -0.111 0.911360
## Gross             -3.093e-09  7.449e-10  -4.153 3.29e-05 ***
## Date               6.210e-01  1.606e-01   3.866 0.000111 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4025.8  on 3286  degrees of freedom
## Residual deviance: 1168.5  on 3273  degrees of freedom
## AIC: 1196.5
##
## Number of Fisher Scoring iterations: 8
```

```r
#Running null model with intercept only

null_model = glm(imdbRating ~ 1, family = "binomial", data = lr_train_temp)
summary(null_model)
```

```
##
## Call:
## glm(formula = imdbRating ~ 1, family = "binomial", data = lr_train_temp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.8476  -0.8476  -0.8476   1.5479   1.5479
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8388     0.0380  -22.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4025.8  on 3286  degrees of freedom
## Residual deviance: 4025.8  on 3286  degrees of freedom
## AIC: 4027.8
```

```
##
## Number of Fisher Scoring iterations: 4
```

```
step(null_model, list(upper = full_model), direction = 'forward')
```

```
## Start:  AIC=4027.77
## imdbRating ~ 1
##
##                     Df Deviance    AIC
## + tomatoUserMeter    1   1497.0 1501.0
## + tomatoUserRating   1   2177.0 2181.0
## + tomatoRating       1   2230.4 2234.4
## + tomatoMeter        1   2451.7 2455.7
## + tomatoFresh        1   3382.9 3386.9
## + imdbVotes          1   3523.1 3527.1
## + tomatoRotten       1   3543.7 3547.7
## + Runtime            1   3711.9 3715.9
## + tomatoReviews      1   3911.9 3915.9
## + Year               1   3920.6 3924.6
## + Date               1   3922.7 3926.7
## + Gross              1   3976.4 3980.4
## + tomatoUserReviews  1   4018.8 4022.8
## + Budget             1   4021.9 4025.9
## <none>                   4025.8 4027.8
##
## Step:  AIC=1500.98
## imdbRating ~ tomatoUserMeter
##
##                     Df Deviance    AIC
## + tomatoRating       1   1305.6 1311.6
## + tomatoMeter        1   1360.6 1366.6
## + tomatoFresh        1   1369.6 1375.6
## + tomatoReviews      1   1412.6 1418.6
## + imdbVotes          1   1447.9 1453.9
## + Runtime            1   1467.1 1473.1
## + tomatoUserReviews  1   1481.3 1487.3
## + tomatoUserRating   1   1489.0 1495.0
## + Budget             1   1490.9 1496.9
## + Gross              1   1494.5 1500.5
## <none>                   1497.0 1501.0
## + Date               1   1495.7 1501.7
## + Year               1   1495.9 1501.9
## + tomatoRotten       1   1497.0 1503.0
##
## Step:  AIC=1311.58
## imdbRating ~ tomatoUserMeter + tomatoRating
##
##                     Df Deviance    AIC
## + tomatoRotten       1   1249.5 1257.5
## + tomatoReviews      1   1265.4 1273.4
## + imdbVotes          1   1271.0 1279.0
## + Runtime            1   1274.4 1282.4
## + tomatoFresh        1   1279.8 1287.8
## + tomatoMeter        1   1288.0 1296.0
```

```
## + tomatoUserReviews  1   1292.5 1300.5
## + tomatoUserRating   1   1295.1 1303.1
## + Budget             1   1297.2 1305.2
## + Date               1   1300.0 1308.0
## + Year               1   1300.2 1308.2
## <none>                   1305.6 1311.6
## + Gross              1   1304.3 1312.3
##
## Step:  AIC=1257.52
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten
##
##                     Df Deviance    AIC
## + Runtime            1   1235.5 1245.5
## + tomatoUserReviews  1   1239.8 1249.8
## + Gross              1   1242.9 1252.9
## + tomatoUserRating   1   1244.7 1254.7
## + imdbVotes          1   1245.9 1255.9
## + tomatoMeter        1   1246.3 1256.3
## + Budget             1   1246.7 1256.7
## <none>                   1249.5 1257.5
## + Year               1   1249.2 1259.2
## + Date               1   1249.3 1259.3
## + tomatoReviews      1   1249.5 1259.5
## + tomatoFresh        1   1249.5 1259.5
##
## Step:  AIC=1245.55
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime
##
##                     Df Deviance    AIC
## + Gross              1   1226.0 1238.0
## + tomatoUserReviews  1   1226.5 1238.5
## + Budget             1   1228.6 1240.6
## + tomatoUserRating   1   1229.6 1241.6
## + imdbVotes          1   1233.3 1245.3
## <none>                   1235.5 1245.5
## + tomatoMeter        1   1233.9 1245.9
## + Date               1   1235.5 1247.5
## + Year               1   1235.5 1247.5
## + tomatoReviews      1   1235.5 1247.5
## + tomatoFresh        1   1235.5 1247.5
##
## Step:  AIC=1238.05
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime + Gross
##
##                     Df Deviance    AIC
## + imdbVotes          1   1208.0 1222.0
## + tomatoUserReviews  1   1214.0 1228.0
## + tomatoUserRating   1   1219.0 1233.0
## <none>                   1226.0 1238.0
## + tomatoMeter        1   1225.1 1239.1
## + tomatoReviews      1   1225.7 1239.7
## + tomatoFresh        1   1225.7 1239.7
```

```
## + Budget             1   1225.8 1239.8
## + Date               1   1226.0 1240.0
## + Year               1   1226.0 1240.0
##
## Step:  AIC=1222
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime + Gross + imdbVotes
##
##                     Df Deviance    AIC
## + tomatoUserRating   1   1197.9 1213.9
## + tomatoUserReviews  1   1199.9 1215.9
## <none>                   1208.0 1222.0
## + tomatoMeter        1   1206.3 1222.3
## + Budget             1   1207.4 1223.4
## + tomatoReviews      1   1207.5 1223.5
## + tomatoFresh        1   1207.5 1223.5
## + Date               1   1207.9 1223.9
## + Year               1   1208.0 1224.0
##
## Step:  AIC=1213.91
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime + Gross + imdbVotes + tomatoUserRating
##
##                     Df Deviance    AIC
## + tomatoUserReviews  1   1187.6 1205.6
## + tomatoMeter        1   1195.8 1213.8
## <none>                   1197.9 1213.9
## + tomatoReviews      1   1196.5 1214.5
## + tomatoFresh        1   1196.5 1214.5
## + Year               1   1196.5 1214.5
## + Date               1   1196.9 1214.9
## + Budget             1   1197.5 1215.5
##
## Step:  AIC=1205.62
## imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime + Gross + imdbVotes + tomatoUserRating + tomatoUserReviews
##
##                 Df Deviance    AIC
## <none>               1187.6 1205.6
## + tomatoMeter    1   1185.7 1205.7
## + tomatoReviews  1   1185.9 1205.9
## + tomatoFresh    1   1185.9 1205.9
## + Year           1   1186.1 1206.1
## + Date           1   1186.5 1206.5
## + Budget         1   1187.5 1207.5


##
## Call:  glm(formula = imdbRating ~ tomatoUserMeter + tomatoRating + tomatoRotten +
##     Runtime + Gross + imdbVotes + tomatoUserRating + tomatoUserReviews,
##     family = "binomial", data = lr_train_temp)
##
## Coefficients:
##       (Intercept)    tomatoUserMeter       tomatoRating       tomatoRotten
##        -2.958e+01          1.934e-01          1.253e+00          1.580e-02
```

```
##           Runtime              Gross          imdbVotes   tomatoUserRating
##         1.597e-02         -3.138e-09          4.503e-06          1.311e+00
## tomatoUserReviews
##         6.591e-08
##
## Degrees of Freedom: 3286 Total (i.e. Null);  3278 Residual
## Null Deviance:       4026
## Residual Deviance: 1188  AIC: 1206
```

```
fitted_model = fitted(full_model)
table(fitted_model>0.5, lr_train_temp$imdbRating)
```

```
##
##            0    1
##   FALSE 2171  123
##   TRUE   124  869
```

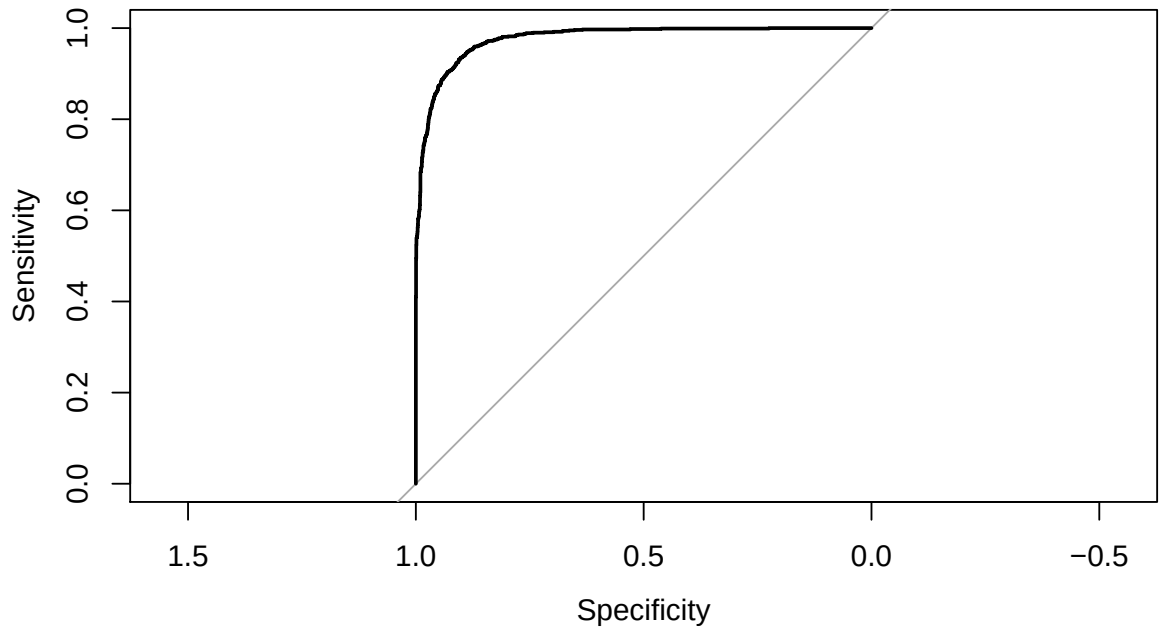
```
table(fitted_model>0.7,lr_train_temp$imdbRating) #selecting 0.7 as our threshold for Y
```

```
##
##            0    1
##   FALSE 2241  232
##   TRUE    54  760
```

```
roc_data = data.frame(fit = fitted_model, obs = lr_train_temp$imdbRating)
my_roc = roc(roc_data$obs ~ roc_data$fit, plot = FALSE)

cat("AUC", toString(auc(my_roc)))
```

```
## AUC 0.976879963454916
```

```
#png(file="C:/Users/uzair/Desktop/Stanford/MS&E 226/Project/ROC.png", type = "cairo")
plot(my_roc)
```



###Code below is a different approach to classification using CV

```
# 0-1 loss cost function
cost_function = function(y, y_hat) {
    mean(as.integer(y_hat > 0.5) != y)
}
```

```
#### Best AIC model####

lr = glm(formula = imdbRating ~ .,
         family = "binomial",
         data = lr_train_temp)
print(summary(lr))
```

```
##
## Call:
## glm(formula = imdbRating ~ ., family = "binomial", data = lr_train_temp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6521  -0.1649  -0.0155   0.0806   4.4767
##
## Coefficients: (1 not defined because of singularities)
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.290e+00  1.690e+01  -0.254 0.799627
## Year              -6.344e-01  1.628e-01  -3.898 9.69e-05 ***
## Runtime            1.599e-02  3.994e-03   4.003 6.24e-05 ***
## imdbVotes          5.023e-06  1.234e-06   4.071 4.69e-05 ***
## tomatoMeter       -1.827e-02  1.232e-02  -1.483 0.138137
## tomatoRating       1.623e+00  2.636e-01   6.155 7.50e-10 ***
```

```
## tomatoReviews      1.836e-02  4.667e-03   3.933 8.38e-05 ***
## tomatoFresh       -1.834e-02  6.132e-03  -2.990 0.002786 **
## tomatoRotten              NA         NA      NA       NA
## tomatoUserMeter    1.936e-01  1.356e-02  14.274  < 2e-16 ***
## tomatoUserRating   1.400e+00  4.126e-01   3.393 0.000692 ***
## tomatoUserReviews  6.655e-08  1.937e-08   3.435 0.000592 ***
## Budget            -3.562e-10  3.200e-09  -0.111 0.911360
## Gross             -3.093e-09  7.449e-10  -4.153 3.29e-05 ***
## Date               6.210e-01  1.606e-01   3.866 0.000111 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4025.8  on 3286  degrees of freedom
## Residual deviance: 1168.5  on 3273  degrees of freedom
## AIC: 1196.5
##
## Number of Fisher Scoring iterations: 8
```

```
###Calculating Error

lr.error = mean(as.integer(lr$fitted.values > 0.5) != lr_train_temp$imdbRating)
print(lr.error)
```

```
## [1] 0.07514451
```

```
# Cross validation for test error estimation

lr.cv = cv.glm(data=lr_train_temp, glmfit=lr, cost=cost_function, K=10)
lr.cv.error = lr.cv$delta[1]
print(lr.cv.error)
```

```
## [1] 0.07727411
```

```
###Calculating accuracy on training set
# fitted.results.train<-predict(lr, lr_train_temp)
# fitted.results.train<-ifelse(fitted.results>0.5,1,0)
# misClasificError.train<-mean(fitted.results != lr_train_temp$imdbRating)
# print(paste('Accuacy',1-misClasificError.train))
```

```
#Running Interaction model

lr.interaction = glm(imdbRating~. + .:., family = "binomial", data = lr_train_temp)
print(summary(lr.interaction))
```

```
##
## Call:
## glm(formula = imdbRating ~ . + .:., family = "binomial", data = lr_train_temp)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
```

```
##  -8.49    0.00    0.00    0.00    8.49
##
## Coefficients: (12 not defined because of singularities)
##                                    Estimate Std. Error   z value Pr(>|z|)
## (Intercept)                       1.157e+18  3.026e+10  38244872   <2e-16 ***
## Year                              4.739e+14  4.898e+07   9675670   <2e-16 ***
## Runtime                          -1.854e+14  1.042e+07 -17787035   <2e-16 ***
## imdbVotes                        -6.678e+10  4.362e+03 -15308509   <2e-16 ***
## tomatoMeter                      -3.372e+14  2.985e+07 -11297714   <2e-16 ***
## tomatoRating                     -5.521e+15  6.117e+08  -9026281   <2e-16 ***
## tomatoReviews                    -1.506e+15  2.075e+07 -72580924   <2e-16 ***
## tomatoFresh                       1.858e+15  3.009e+07  61742295   <2e-16 ***
## tomatoRotten                             NA         NA        NA       NA
## tomatoUserMeter                  -8.524e+14  4.029e+07 -21156023   <2e-16 ***
## tomatoUserRating                  8.784e+16  1.657e+09  53018225   <2e-16 ***
## tomatoUserReviews                 3.550e+10  3.967e+02  89484177   <2e-16 ***
## Budget                            6.441e+07  2.177e+01   2958270   <2e-16 ***
## Gross                             2.619e+08  4.845e+00  54051240   <2e-16 ***
## Date                             -1.734e+15  4.908e+07 -35331350   <2e-16 ***
## Year:Runtime                     -5.861e+12  1.156e+05 -50686816   <2e-16 ***
## Year:imdbVotes                   -4.068e+09  6.633e+01 -61335510   <2e-16 ***
## Year:tomatoMeter                 -3.330e+12  3.538e+05  -9412366   <2e-16 ***
## Year:tomatoRating                 1.259e+14  6.969e+06  18069167   <2e-16 ***
## Year:tomatoReviews               -1.753e+12  1.898e+05  -9239463   <2e-16 ***
## Year:tomatoFresh                  4.444e+12  2.676e+05  16610045   <2e-16 ***
## Year:tomatoRotten                        NA         NA        NA       NA
## Year:tomatoUserMeter             -3.656e+12  4.625e+05  -7904520   <2e-16 ***
## Year:tomatoUserRating            -2.267e+14  1.773e+07 -12784529   <2e-16 ***
## Year:tomatoUserReviews            1.028e+09  5.733e+01  17924005   <2e-16 ***
## Year:Budget                      -1.390e+07  2.627e-01 -52920610   <2e-16 ***
## Year:Gross                        2.193e+06  1.018e-01  21547382   <2e-16 ***
## Year:Date                         3.393e+11  7.318e+03  46365407   <2e-16 ***
## Runtime:imdbVotes                 3.405e+07  6.350e-01  53618180   <2e-16 ***
## Runtime:tomatoMeter              -3.593e+11  1.033e+04 -34779512   <2e-16 ***
## Runtime:tomatoRating              5.447e+12  1.824e+05  29869599   <2e-16 ***
## Runtime:tomatoReviews             4.616e+10  3.367e+03  13709149   <2e-16 ***
## Runtime:tomatoFresh              -7.981e+09  4.267e+03  -1870417   <2e-16 ***
## Runtime:tomatoRotten                     NA         NA        NA       NA
## Runtime:tomatoUserMeter           7.928e+10  9.318e+03   8508536   <2e-16 ***
## Runtime:tomatoUserRating          2.313e+12  3.657e+05   6325307   <2e-16 ***
## Runtime:tomatoUserReviews         2.907e+05  2.490e-02  11675118   <2e-16 ***
## Runtime:Budget                   -7.409e+04  2.621e-03 -28263199   <2e-16 ***
## Runtime:Gross                    -1.742e+04  7.077e-04 -24613058   <2e-16 ***
## Runtime:Date                      5.940e+12  1.144e+05  51921590   <2e-16 ***
## imdbVotes:tomatoMeter            -2.262e+08  3.137e+00 -72099198   <2e-16 ***
## imdbVotes:tomatoRating            1.687e+09  3.940e+01  42816859   <2e-16 ***
## imdbVotes:tomatoReviews          -6.074e+07  1.026e+00 -59182360   <2e-16 ***
## imdbVotes:tomatoFresh             6.180e+07  1.196e+00  51686516   <2e-16 ***
## imdbVotes:tomatoRotten                   NA         NA        NA       NA
## imdbVotes:tomatoUserMeter        -2.343e+07  2.484e+00  -9432195   <2e-16 ***
## imdbVotes:tomatoUserRating       -1.657e+09  8.123e+01 -20400454   <2e-16 ***
## imdbVotes:tomatoUserReviews       1.391e+02  2.738e-06  50798636   <2e-16 ***
## imdbVotes:Budget                 -5.648e+00  4.624e-07 -12213994   <2e-16 ***
## imdbVotes:Gross                  -1.195e+00  7.892e-08 -15143080   <2e-16 ***
```

```
## imdbVotes:Date                       4.108e+09  6.623e+01  62028878   <2e-16 ***
## tomatoMeter:tomatoRating            -2.954e+12  1.101e+05 -26820430   <2e-16 ***
## tomatoMeter:tomatoReviews           -4.055e+11  3.493e+04 -11608242   <2e-16 ***
## tomatoMeter:tomatoFresh             -2.167e+11  1.134e+04 -19117271   <2e-16 ***
## tomatoMeter:tomatoRotten                    NA         NA        NA       NA
## tomatoMeter:tomatoUserMeter          1.970e+12  2.361e+04  83436140   <2e-16 ***
## tomatoMeter:tomatoUserRating         1.040e+13  9.380e+05  11083480   <2e-16 ***
## tomatoMeter:tomatoUserReviews       -1.018e+07  1.286e-01 -79189251   <2e-16 ***
## tomatoMeter:Budget                  -3.100e+05  9.991e-03 -31026254   <2e-16 ***
## tomatoMeter:Gross                    1.359e+05  2.746e-03  49506259   <2e-16 ***
## tomatoMeter:Date                     3.448e+12  3.508e+05   9829154   <2e-16 ***
## tomatoRating:tomatoReviews          -5.503e+12  1.376e+05 -39998623   <2e-16 ***
## tomatoRating:tomatoFresh             5.683e+12  1.767e+05  32157904   <2e-16 ***
## tomatoRating:tomatoRotten                   NA         NA        NA       NA
## tomatoRating:tomatoUserMeter        -2.007e+13  4.498e+05 -44621123   <2e-16 ***
## tomatoRating:tomatoUserRating       -1.490e+14  1.682e+07  -8855921   <2e-16 ***
## tomatoRating:tomatoUserReviews       8.984e+07  1.818e+00  49416459   <2e-16 ***
## tomatoRating:Budget                  3.975e+06  1.762e-01  22558774   <2e-16 ***
## tomatoRating:Gross                  -6.836e+05  4.586e-02 -14905784   <2e-16 ***
## tomatoRating:Date                   -1.224e+14  6.900e+06 -17736919   <2e-16 ***
## tomatoReviews:tomatoFresh            1.608e+10  1.075e+03  14961096   <2e-16 ***
## tomatoReviews:tomatoRotten          -4.188e+09  1.633e+03  -2565592   <2e-16 ***
## tomatoReviews:tomatoUserMeter        2.849e+11  7.957e+03  35805161   <2e-16 ***
## tomatoReviews:tomatoUserRating       5.295e+12  3.288e+05  16104173   <2e-16 ***
## tomatoReviews:tomatoUserReviews      4.840e+03  3.952e-02    122477   <2e-16 ***
## tomatoReviews:Budget                -3.347e+04  1.919e-03 -17442381   <2e-16 ***
## tomatoReviews:Gross                  3.396e+04  5.758e-04  58982624   <2e-16 ***
## tomatoReviews:Date                   2.494e+12  1.893e+05  13179042   <2e-16 ***
## tomatoFresh:tomatoRotten            -3.859e+09  4.236e+03   -910961   <2e-16 ***
## tomatoFresh:tomatoUserMeter         -2.885e+11  1.163e+04 -24800462   <2e-16 ***
## tomatoFresh:tomatoUserRating        -6.851e+12  4.542e+05 -15083207   <2e-16 ***
## tomatoFresh:tomatoUserReviews        2.356e+06  4.549e-02  51802000   <2e-16 ***
## tomatoFresh:Budget                   7.773e+03  2.894e-03   2686040   <2e-16 ***
## tomatoFresh:Gross                   -2.882e+04  7.880e-04 -36570097   <2e-16 ***
## tomatoFresh:Date                    -5.331e+12  2.665e+05 -20001945   <2e-16 ***
## tomatoRotten:tomatoUserMeter                NA         NA        NA       NA
## tomatoRotten:tomatoUserRating               NA         NA        NA       NA
## tomatoRotten:tomatoUserReviews              NA         NA        NA       NA
## tomatoRotten:Budget                         NA         NA        NA       NA
## tomatoRotten:Gross                          NA         NA        NA       NA
## tomatoRotten:Date                           NA         NA        NA       NA
## tomatoUserMeter:tomatoUserRating    -5.039e+12  2.427e+05 -20763535   <2e-16 ***
## tomatoUserMeter:tomatoUserReviews    3.897e+06  7.327e-02  53187236   <2e-16 ***
## tomatoUserMeter:Budget               1.172e+05  8.479e-03  13827029   <2e-16 ***
## tomatoUserMeter:Gross               -1.511e+04  2.375e-03  -6362456   <2e-16 ***
## tomatoUserMeter:Date                 4.111e+12  4.581e+05   8974478   <2e-16 ***
## tomatoUserRating:tomatoUserReviews -2.489e+08  3.056e+00 -81451058   <2e-16 ***
## tomatoUserRating:Budget              1.941e+06  3.319e-01   5848798   <2e-16 ***
## tomatoUserRating:Gross               4.962e+05  8.876e-02   5590924   <2e-16 ***
## tomatoUserRating:Date                1.831e+14  1.757e+07  10420413   <2e-16 ***
## tomatoUserReviews:Budget             6.555e-01  1.497e-08  43778387   <2e-16 ***
## tomatoUserReviews:Gross             -2.327e-01  3.390e-09 -68636124   <2e-16 ***
## tomatoUserReviews:Date              -1.045e+09  5.733e+01 -18232304   <2e-16 ***
## Budget:Gross                         8.162e-03  1.649e-10  49496769   <2e-16 ***
```

```
## Budget:Date                          1.386e+07  2.618e-01  52960442   <2e-16 ***
## Gross:Date                          -2.328e+06  1.017e-01 -22880143   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:  4025.8  on 3286  degrees of freedom
## Residual deviance: 21337.8  on 3193  degrees of freedom
## AIC: 21526
##
## Number of Fisher Scoring iterations: 25
```

```
###Calculating Error for interaction

lr.error.interaction = mean(as.integer(lr.interaction$fitted.values > 0.5) != lr_train_temp$imdbRating)
print(lr.error.interaction)
```

```
## [1] 0.09005172
```

```
# Cross validation for test error estimation on interaction model

lr.cv.interaction = cv.glm(data=lr_train_temp, glmfit=lr.interaction, cost=cost_function, K=10)
lr.cv.error.interaction = lr.cv.interaction$delta[1]
print(lr.cv.error)
```

```
## [1] 0.07727411
```

```
# # Predictions of the best model and the baseline on the test set
# pred.best = as.numeric(predict(lr, lr_test_temp) >= 0)
# pred.baseline = as.numeric(predict(null_model, lr_test_temp) >= 0)
#
# # Ground truth
# truth = lr_test_temp$imdbRating
#
# # Compute 0-1 loss for best model and baseline
# mean(as.numeric(pred.best != lr_test_temp$imdbRating))
# mean(as.numeric(pred.baseline != lr_test_temp$imdbRating))
#
# # Plot the ROC curve and compute AUC
# prob.baseline = predict(null_model, lr_test_temp, type=c("response"))
# prob.best = predict(lr, lr_test_temp, type=c("response"))
# lr_test_temp$prob.baseline = prob.baseline
# lr_test_temp$prob.best = prob.best
#
# g.baseline <- roc(imdbRating ~ prob.baseline, data = lr_test_temp)
# g.best <- roc(imdbRating ~ prob.best, data = lr_test_temp)
#
# plot(g.baseline, col="red")
# plot(g.best, col="blue", add=TRUE)
# auc(g.best)
# auc(g.baseline)
```